

**AFRL-IF-RS-TR-2003-55**  
**Final Technical Report**  
**March 2003**



# **A NATIONWIDE EXPERIMENTAL MULTI-GIGABIT NETWORK**

**High Speed Connectivity Consortium**

**Sponsored by**  
**Defense Advanced Research Projects Agency**  
**DARPA Order No. G147, J164**

*APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.*

**The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the U.S. Government.**

**AIR FORCE RESEARCH LABORATORY**  
**INFORMATION DIRECTORATE**  
**ROME RESEARCH SITE**  
**ROME, NEW YORK**

This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

AFRL-IF-RS-TR-2003-55 has been reviewed and is approved for publication.

APPROVED:



ROBERT L. KAMINSKI  
Project Engineer

FOR THE DIRECTOR:



WARREN H. DEBANY, Technical Advisor  
Information Grid Division  
Information Directorate

<b>REPORT DOCUMENTATION PAGE</b>			<i>Form Approved</i> <b>OMB No. 074-0188</b>	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing this collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503				
<b>1. AGENCY USE ONLY (Leave blank)</b>		<b>2. REPORT DATE</b> MARCH 2003	<b>3. REPORT TYPE AND DATES COVERED</b> Final Sep 98 – Dec 02	
<b>4. TITLE AND SUBTITLE</b> A NATIONWIDE EXPERIMENTAL MULTI-GIGABIT NETWORK			<b>5. FUNDING NUMBERS</b> C - F30602-98-2-0193 PE - 62301E PR - G147 TA - 00 WU - 01	
<b>6. AUTHOR(S)</b> Raj Reddy, et al				
<b>7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)</b> High Speed Connectivity Consortium 123 University Place Pittsburgh Pennsylvania 15213			<b>8. PERFORMING ORGANIZATION REPORT NUMBER</b>	
<b>9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)</b> Defense Advanced Research Projects Agency AFRL/IFGC 3701 North Fairfax Drive 525 Brooks Road Arlington Virginia 22203-1714 Rome New York 13441-4505			<b>10. SPONSORING / MONITORING AGENCY REPORT NUMBER</b>  AFRL-IF-RS-TR-2003-55	
<b>11. SUPPLEMENTARY NOTES</b>  AFRL Project Engineer: Robert L. Kaminski/IFGC/(315) 330-1865/ Robert.Kaminski@rl.af.mil				
<b>12a. DISTRIBUTION / AVAILABILITY STATEMENT</b> APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.				<b>12b. DISTRIBUTION CODE</b>
<b>13. ABSTRACT (Maximum 200 Words)</b> <p>The High Speed Connectivity Consortium (HSCC) created a nation-wide multi-gigabit network, capable of gigabit connections to end user sites, using fiber optic links at OC-48 rates. The consortium provided high-speed access to the network with consumption-based pricing for affordability.</p> <p>The network backbone was provided by Qwest using their national network. Local access was provided by various sources such as power utilities, Competitive local exchange carriers, and other Right-of-Way owners.</p> <p>The network provided high speed connectivity for research in networking architectures, high bandwidth applications, and protocol research.</p> <p>Specifically, the Matissee Project, a joint collaboration between UC Berkeley, LBNL, CMU, MIT, CNRI and USC/ISI utilized the network for remote MEMS design, fabrication and testing/experiments. The network enabled research into why host systems and the TCP protocols have so much difficulty achieving high performance when operating across high bandwidth delay product networks. The network also enabled research and testing into the distribution of Uncompressed HTDV across wide area networks.</p>				
<b>14. SUBJECT TERMS</b> Optical Networking, Gigabit Networks, Network Protocols, Network Architectures			<b>15. NUMBER OF PAGES</b> 127	
			<b>16. PRICE CODE</b>	
<b>17. SECURITY CLASSIFICATION OF REPORT</b>  UNCLASSIFIED	<b>18. SECURITY CLASSIFICATION OF THIS PAGE</b>  UNCLASSIFIED	<b>19. SECURITY CLASSIFICATION OF ABSTRACT</b>  UNCLASSIFIED	<b>20. LIMITATION OF ABSTRACT</b>  UL	

## Table of Contents

Introduction.....	1
Technical Report.....	1
Distributed Science Classroom Experiments.....	2
METACARTA subcontract .....	2
Technical Highlights.....	3
Publications.....	6
APPENDIX A PUBLICATIONS .....	8
IMPLEMENTING CONGESTION CONTROL IN THE REAL WORLD	9
Large Group Teleconferencing: Techniques and Considerations	13
Yima: A Second-Generation Continuous Media Server	31
Experiments with Delivery of HDTV over IP Networks	40
Applied Techniques for High Bandwidth Data Transfers across Wide Area Networks	52
Enabling Network-Aware Applications	61
Using High-Speed WANs and Network Data Caches to Enable Remote and Distributed Visualization	69
METACARTA Geographic Text Search	92

## **Acknowledgments**

The Principal Investigator wishes to acknowledge Terry Gibbons and Tom Lehman of the University of Southern California, Information Sciences Institute (East), Arlington, Virginia for their collaborative efforts.

## **Introduction**

This final report is being submitted by High Speed Connectivity Consortium (HSCC), for work performed under Award Agreement No. F30602-98-2-0193, for the period September 18, 1998 through December 31, 2002.

## **Technical Report**

Beginning September 18, 1998, HSCC planned to have five sites operational at OC 48 rates. The contract was subsequently modified to have four sites and a distributed classroom experiment which uses the high bandwidth. A geographical index of the web experiment was added as a subcontract to Meta-Carda in July 2001.

### **1. Los Angeles, CA - NTON Network**

This network connection was operational at OC 48 rates between October 15, 1999 and November 04, 2002. It linked NTON (National Testbed for Optical Networking) Network to the HSCC backbone via a direct in-building connection on Wilshire Boulevard in LA. Using HSCC network NTON provided connectivity for Lawrence Livermore Laboratory (LL), NASA, Ames and SRI.

### **2. Washington, DC - ATD Network**

This network connection was in operation at OC 48 rates between October 25, 1999 and November 04, 2002. It linked the DARPA supported ATD (Advanced Technology Demonstration) network to the HSCC backbone via an OC-48 line from ISI-East.

### **3. Pittsburgh, PA - Pittsburgh Super Computer Center**

This network connection became operational at OC 12 rates since December 1, 1999. It was upgraded to OC 48 rates in January, 2001 and was in operation till Nov 04, 2002. It linked Carnegie Mellon University to the HSCC backbone via an intermediate connection at the Pittsburgh Supercomputer Center (PSC).

### **4. Seattle, WA- Pacific Northwest Network**

This network connection was in place since December 15, 1999. However, this link was terminated as of June 30, 2000 due to the merger between Qwest and US West communications.

HSCC attempted to include Argonne National Laboratory in Argonne, IL as the new fourth node for the last year of operation. However due to the difficulties in obtaining local access fiber this option did not become operational in time.

## **Distributed Science Classroom Experiments**

In December 2000, HSCC requested and received a modification to change the work statement for this project. Accordingly, HSCC have provided partial funding for distributed Classroom experiments to four nationally recognized Universities.

Brown University  
Carnegie Mellon University  
University of California at Berkeley  
University of Washington at Seattle

Experiments are ongoing and will continue beyond the scope of this contract.

## **METACARTA subcontract**

In July 2001, HSCC requested and received additional funding for the METACARTA project. These funds were provided to create software to support the following activities

- Generate a prototype geographic index of web pages.
- Geographic structuring of non-relational information.
- Location specific applications that require surrounding information for analysis and planning.
- Wireless applications that require proximity- sensitive searching.
- Platform for organizing data from instruments, sensors, and communications.

The MetaCarta final report can be read in its entirety at URL:

[http://www.hsccl.net/MetaCarta\\_GTS\\_Appliance.ppt](http://www.hsccl.net/MetaCarta_GTS_Appliance.ppt).

## Technical Highlights

SuperNet/HSCC continued its support for multiple research programs and demonstrations, which are described in detail at this web URL: <http://www.ngi-supernet.org/experiments.html>.

Various programs under NGI-Supernet were able to utilized the bandwidth which was provided by HSCC:

- [Matisse](#)
- [BOSSNET](#)
- [Gigabit To The Desktop](#)
- [Gigabit Rate IP Security](#)
- [Secure Network Toolbox \(Secure Network Monitoring and Management Infrastructure\)](#)
- [High Performance Local Area Networks \(10-40Gb/s\)](#)
- [NGI Multicast Applications and Architecture \(NMAA\)](#)
- [Uncompressed High Definition Television \(HDTV\) over IP](#)
- [Access Grid \(AG\)](#)
- [Collaborative Advanced Interagency Research Network \(CAIRN\) Experiments](#)



The List of the research programs, which utilized HSCC bandwidth, is given below:

- TCP performance across high bandwidth-delay product networks
- Remote Media Immersion (RMI)
- IMSC's Remote Media Immersion (RMI)
- Integrated Media Systems Center (IMSC)
- Matisse
- Distributed-Parallel Storage System (DPSS)
- Gigabit To The Desktop
- Gigabit Rate IP Security
- Secure Network Toolbox (Secure Network Monitoring and Management Infrastructure)
- Collaborative, Operational Virtual Exploitation Team (COVET)
- NGI Multicast Applications and Architecture (NMAA)
- Uncompressed High Definition Television (HDTV) over IP
- Access Grid (AG)
- Collaborative Advanced Interagency Research Network (CAIRN) Experiments
- X-Bone (Automated Overlay Network Deployment)
- Active Networks Backbone (ABone)
- National Internet Measurement Infrastructure (NIMI)
- Multicast-based Inference of Network-internal Characteristics (MINC)
- Secure Border Gateway Protocol (S-BGP)
- Border Gateway Multicast Protocol (BGMP)
- Network Time Synchronization Project (NTSP)
- Reliable Multicast Performance
- DNS Security (DNSSEC) in CAIRN
- Secure Network Toolbox (SNMPv3, SSL, SSH)
- SNMPv3 in CAIRN
- Fault-Tolerant Networking Through Intrusion Identification and Secure Compartments (FNIISC)
- Fault-Tolerant Mesh of Trust Applied to DNSSEC (FMESHD)
- Bro: A System for Detecting Network Intruders in Real-Time
- Realizing Adaptive Distributive Internet Operations on ACTIVE Networks (RADIOACTIVE)
- Secure Conferencing Access with Multicast Protocols for the Internet (SCAMPI)

The List of experiments and demonstrations, which utilized HSCC bandwidth, is given below:

- ACCESS Facility Demos
- Stereoscopic Rendered Images and Video Streaming with Real-time
- Compression Methods (Internet2 and Super Net infrastructure)
- Telepresence in the Operating Room Utilizing IP Video (Internet2 and Super Net infrastructure)
- Super Computing 2000
- Accelerated Strategic Computing Initiative (ASCI)
- VisaPult: Image Based Rendering Assisted Volume Rendering - SC2000
- Network Challenge Winner
- Cal Tech Particle Physics Using Globus
- Data Management Infrastructure for Climate Modeling Research (Striped FTP)-SC2000 Network Challenge Winner
- Stanford Linear Accelerator Center (SLAC)
- NASA Digital Sky Demo
- Digital Amphitheater
- Digital Earth
- Land Speed Record
- Internet2 Land Speed Record
- UW-ISle High Bandwidth Tests
- UW-ISle Internet HDTV Tests
- Super Computing 2001
- TeleImmersion

## Publications

As a result of HSCC's collaborative efforts the following articles were published.

- Retransmission-Based Error Control in a Many-to-Many Client-Server Environment. Roger Zimmermann, Kun Fu, Nitin Nahata, and Cyrus Shahabi. Accepted for presentation at the SPIE Conference on Multimedia Computing and Networking 2003(MMCN 2003), Santa Clara, California, January 29-31, 2003.
- Ladan Gharai & Colin Perkins, Implementing Congestion Control in the Real World, Proceedings of the IEEE International Conference on Multimedia and Expo, Lausanne, Switzerland, August 2002.  
<http://www.east.isi.edu/projects/NMAA/hdtv/publications/icme2002.pdf>
- Ladan Gharai, Colin Perkins & Allison Mankin, Large Group Teleconferencing: Techniques and Considerations, Proceedings of the 3rd International Conference on Internet Computing, Las Vegas, June 2002.  
<http://csperkins.org/publications/ic2002.pdf>
- Christian Rembe, Rishi Kant, Michael P. Young, Richard S. Muller, "Network-connected MEMS-measuring system for high-speed data transfer to CAD and simulation tools," Conference on Vibration Measurements by Laser Techniques," Italian Assn. for Laser Velocimetry, Ancona, Italy, 18-21 June 2002
- Yima: A Second Generation Continuous Media Server. Cyrus Shahabi, Roger Zimmermann, Kun Fu, and Shu-Yuen Didi Yao. Published in the IEEE Computer magazine, June 2002, pp. 56-64.  
<http://idefix.usc.edu/pubs/IEEEComp.pdf>
- On Internet of the Future, Surfers May Almost Feel the Spray, New York Times Article, May 9, 2002. Article about RMI.  
<http://idefix.usc.edu/pubs/NYTimes-RMI.pdf>
- Colin Perkins, Ladan Gharai, Tom Lehman & Allison Mankin, Experiments with delivery of HDTV over IP Networks, Proceedings of the 12th International Packet Video Workshop, Pittsburgh, April 2002.  
<http://www.east.isi.edu/projects/NMAA/hdtv/publications/pv2002.pdf>
- J. Lee, D. Gunter, B. Tierney, W. Allock, J. Bester, J. Bresnahan, S. Tuecke, " Applied Techniques for High Bandwidth Data Transfers across Wide Area Networks", Proceedings of Computers in High Energy Physics 2001 (CHEP 2001), Beijing China, LBNL-46269.  
[http://www-didc.lbl.gov/papers/dpss\\_and\\_gridftp.pdf](http://www-didc.lbl.gov/papers/dpss_and_gridftp.pdf)

- B. Tierney, D. Gunter, J. Lee, M. Stoufer, "Enabling Network-Aware Applications", Proceedings of the 10th IEEE Symposium on High Performance Distributed Computing (HPDC-10), August 2001, LBNL-47611. <http://www.didc.lbl.gov/papers/Enable.HPDC01.pdf>
- W. Bethel, Tierney, B., Lee, J., Gunter, D., Lau, S., "Using High-Speed WANs and Network Data Caches to Enable Remote and Distributed Visualization", Proceeding of the IEEE Supercomputing 2000 Conference, Nov. 2000. LBNL-45365. <http://www.didc.lbl.gov/papers/visapult-sc00.pdf>
- W. Bethel, B. Tierney, J. Lee, D. Gunter, S. Lau, "Using High-Speed WANs and Network Data Caches to Enable Remote and Distributed Visualization," in Proceedings of SC00, November 2000. 2000/LBNL-45365-VisapultSC00.pdf (LBNL 45365).  
<http://www-vis.lbl.gov/Publications/2000/LBNL-45365-VisapultSC00.pdf>

The MetaCarta final report can be read in its entirety at URL:

[http://www.hsccl.net/MetaCarta\\_GTS\\_Appliance.ppt](http://www.hsccl.net/MetaCarta_GTS_Appliance.ppt)

Print outs of these hyper-linked publications are attached in Appendix A.

## **APPENDIX A PUBLICATIONS**

# IMPLEMENTING CONGESTION CONTROL IN THE REAL WORLD

*Ladan Gharai   Colin Perkins*

University of Southern California  
Information Sciences Institute

## ABSTRACT

It is well known that congestion control is a key issue for the safe deployment of multimedia applications over IP. We describe our initial experiences implementing TCP-friendly congestion control in a system designed to deliver HDTV content over IP. In particular we discuss the effects of packet reordering on the calculated throughput, and highlight the problems this can pose for high-rate applications.

## 1. INTRODUCTION

Given the proliferation of high speed networks and multimedia applications, it is becoming increasingly important to consider congestion control. This is especially critical for applications with unusual bandwidth requirements, due to their potential to disrupt existing network traffic.

An example of the emerging class of ultra-high rate multimedia applications might be delivery of gigabit rate high definition television (HDTV) signals over IP networks. We have implemented such a system [7], at a constant data rate of 850 Mbps, and have experience of the problems such high rate traffic can cause. To make this application safe for use outside carefully controlled testbeds, we desired to implement congestion control. This paper describes our initial experiences with TCP-friendly rate control of this application.

The paper is organized as follows. Section 2 describes the demonstrator system, and outlines algorithms for multimedia congestion control. Section 3 describes our implementation, while Sections 4 and 5 discuss experimental setup and results. The lessons learnt from our experiment are described in section 6, along with directions for further work. Finally, Section 7 concludes the paper.

## 2. BACKGROUND

In previous work, we developed a prototype telepresence system that uses HDTV equipment to provide very high

quality telepresence over IP networks [7]. The system runs at rates of approximately 850 Mbps, delivering 1280x720 pixel video at 60 frames-per-second in 24-bit YUV color. It is implemented with off-the-shelf components: a PC-based server running Linux, with HDTV I/O and gigabit Ethernet cards. It uses standard RTP over UDP/IP network transfer protocols [8, 4].

Our wide area tests with this system proved the viability of transporting high bandwidth video streams over IP. However, they also highlighted a severe limitation: due to the lack of congestion control our tests could only be conducted with permission, and careful monitoring, from the network operations staff, so as to ensure that such a high-rate non-congestion controlled stream did not adversely affect other traffic on the network.

In order for multimedia traffic and TCP/IP flows to co-exist and receive a fair share of available bandwidth, the non-TCP traffic must be TCP friendly. A TCP friendly flow will fairly share bandwidth with other flows, while judiciously seeking free bandwidth. It has been shown that, for a saturated steady state TCP sender, throughput is proportional to inverse of the square root of the packet loss rate,  $p$  [5]. This is known as the TCP-friendly equation, and it provides an upper bound on the steady state throughput  $T$ , for packet size  $S$ , round trip time  $R$ , retransmission timeout  $t_{RTO} \approx 4R$  and the steady state loss event rate  $p$ , such that:

$$T = \frac{S}{R\sqrt{\frac{2p}{3}} + t_{RTO}(3\sqrt{\frac{3p}{8}})p(1 + 32p^2)} \quad (1)$$

Utilizing the TCP-friendly equation has resulted in a class of equation based congestion control schemes, such as the TCP friendly rate control (TFRC) protocol [3]. The basic concept is to regulate throughput using equation 1, guaranteeing that the flow is TCP-friendly. Once a sender is aware of the loss event rate  $p$  and the round trip time  $R$ , it can compute its fair share of bandwidth and adjust its sending rate accordingly. Damping is applied, to ensure that the rate of adaptation is smoother than TCP, while maintaining long-term fairness. The dynamics of TFRC, and its interaction with TCP, are described in [3].

### 3. DESIGN AND IMPLEMENTATION

TCP friendly rate control relies on the sender being able to adjust its sending rate according to the amount of loss the flow is experiencing. In TFRC, loss is measured as a *loss event fraction* by the receiver. TFRC distinguishes between loss fraction and loss event fraction, to better emulate TCP. Loss event fraction measures the fraction of loss occurring more than one round trip time (*RTT*) apart. In other words, once an initial loss occurs, any other following loss within a *RTT* is ignored. This closely mimics most TCP variants.

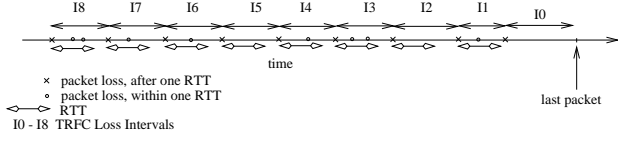


Figure 1: TFRC Loss Intervals.

Handling of loss intervals in TFRC is shown in Figure 1. TFRC recommends the use of  $N = 8$  intervals, however as seen in Figure 1,  $N + 1$  intervals are actually maintained. To compute the average loss interval, TFRC chooses the maximum of the values of  $\sum_{n=0}^7 I_n$  and  $\sum_{n=1}^8 I_n$ . Therefore, if the interval since the last packet loss event,  $I_0$ , is large, it is accounted for in the computation of the loss event rate, helping TFRC increase its sending rate in the absence of loss.

To implement TFRC, the following two feedback loops are needed: first, the sender must periodically send perceived RTT to the receiver, thereby allowing the receiver to compute the loss event rate,  $p$ . Secondly, the receiver must send the computed loss event rate,  $p$ , back to the sender. Figure 2 illustrates the process.

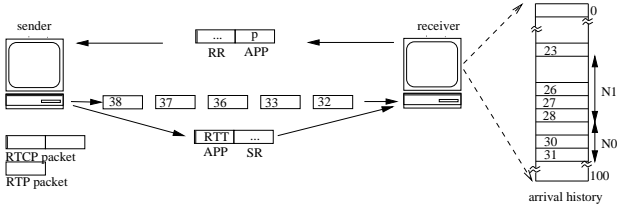


Figure 2: TFRC feedback loops implemented in RTCP.

Our implementation uses RTP over UDP/IP transport. RTP provides feedback using the RTP Control Protocol, RTCP. At regular intervals, implementations generate Receiver Report (RR) or Sender Report (SR) packets, providing reception quality feedback and support for lip-synchronization. Application specific feedback is supported using APP packets, that are piggy-backed at regular intervals with RR or SR

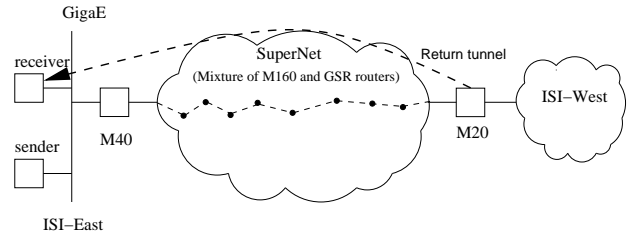


Figure 3: The network used in our tests.

packets. In our implementation, each time the sender generates a sender report it also sends the *RTT* to the receiver in an APP packet. Likewise, when the receiver sends back a receiver report it also includes an APP packet with the latest computation on the loss event rate,  $p$ .

### 4. EXPERIMENTAL SETUP

To test our system, we need a wide-area network capable of supporting high rate UDP flows. Several such networks have become available recently, including Internet2 and the DARPA SuperNet testbed. We report on tests conducted using SuperNet (previous experiments have used Internet2).

The SuperNet testbed comprises several research networks, connected using a cross-country overlay on a commercial ISP network. The individual research networks are multi-gigabit capacity, and the overlay is intended to support gigabit rate applications. In practice, the capacity of the overlay network varies with the load on the underlying network.

The network path we tested is shown in Figure 3. The wide area path from ISI East in Arlington, VA, to ISI West in Los Angeles is nine IP hops. We configured a tunnel to return traffic from the router in LA, looping traffic back to our laboratory. This allows us to display the results, and gives a network path with 10 logical – 18 actual – hops and a 132ms round trip time.

The sender and receiver are Dell PowerEdge 2500 servers with dual 1.2GHz Pentium III processors, running Linux 2.4.2. They are equipped with 3Com 3c985 gigabit Ethernet and DVS HDstationOEM HDTV interface cards. We capture live HDTV content, packetize and transmit RTP packets destined for the tunnel interface of the receiver. The routing is such that the packets traverse the network before returning though the tunnel to the receiver, where they are depacketized and displayed. The full rate of the system is 850 Mbps, although it can adapt by sending at reduced frame rate.

When the underlying network is lightly loaded, we have consistently been able to run cross-country HDTV-over-IP

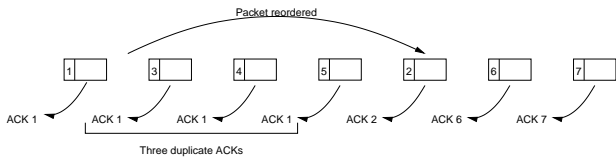


Figure 4: Packet reordering gives the appearance of loss

at 850 Mbps without packet loss. As the network becomes more loaded, typically during business hours, we see packet loss in our application, indicating congestion in the network.

## 5. EXPERIMENTAL RESULTS

We conducted a number of experiments with our system, both local area and on the wide area network described in Section 4. As expected, the network performance varied: much of the time it was loss free, but there were instances when packet loss was observed, making congestion control necessary.

We are still evaluating the performance of our system in the presence of packet loss, and tuning our congestion control and rate adaptation algorithms. These results are outside the scope of this paper (although we discuss the issues in Section 6). The results we present here reflect our experience when the network was lightly loaded, and loss free.

In the absence of packet loss, we noticed that our congestion control function was suggesting we send at a relatively low rate (and was stable at that rate). This was somewhat unexpected, since TCP performance, and by extension the performance of TFRC congestion control, is driven mostly by packet loss. Indeed, a naive interpretation of equation 1 would say that zero packet loss should result in infinite rate.

That interpretation does not, however, take into account the effects of packet reordering in the network. Experiments showed that some amount, up to 1.3% depending on time of day, of packets were reordered (a value not incompatible with [1, 2, 6]).

Our hypothesis is that reordering causes the congestion control function to return lower-than-expected rates. For example, packets that arrive at least four places out of order would cause TCP to deliver a triple duplicate ACK, giving the appearance of loss (see Figure 4). The analysis behind the TCP-friendly rate control equation [5] reflects this, so TFRC can also be expected to treat reordering as loss.

To validate this hypothesis, we took a closer look at packet reordering and how it effects the computation of the loss event rate. The results shown in Figure 5 plot the evolution

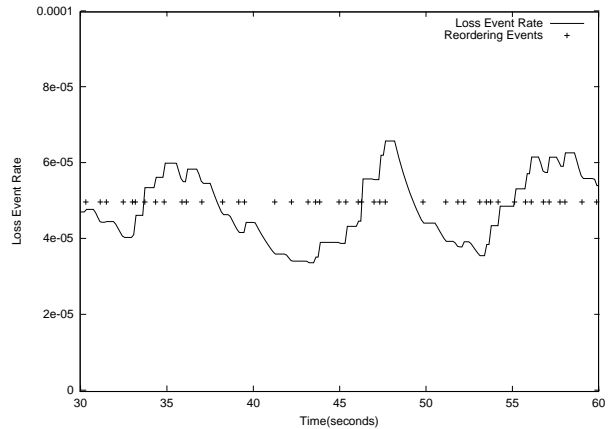


Figure 5: Evolution of Loss Event Rate due to reordering.

of the loss event rate along with reorderings that give the appearance of loss and start a new loss interval,  $I_n$ , as defined by TFRC. It is evident that changes in the loss event rate correlate with the new intervals, demonstrating that significant packet reordering causes TFRC to change its transmission rate.

It is also interesting to note that throughout the graph, when new loss intervals are substantially spaced apart, this results in a gradual reduction in the loss event rate. A good example of this occurs at about second 48 in the graph. As discussed in Section 3, TFRC may or may not include the last interval  $I_0$  in its computation of the average loss intervals. Clearly, around point 48 second in the graph, due to lack of loss,  $I_0$  gradually grows, and this growth correlates to the gradual reduction of the loss event rate.

As an additional validation step, we conducted a number of performance tests with TCP traffic. Although there was not an exact match, we found that – after the hosts were tuned for optimal performance – the Linux TCP stack gave comparable throughput to that predicted by our congestion control function. Our results show the Linux TCP achieving throughput on the order of twice that of our TFRC implementation. This is somewhat more than expected, perhaps due to the use of SACK TCP in Linux which is less sensitive to reordering than the Reno TCP used in the derivation of the TCP-friendly equation, but not unreasonable. Detailed comparison of TCP and TFRC throughput in the presence of reordering is ongoing, but omitted here due to lack of space.

We also note that the fraction of reordered packets we observe appears to be somewhat independent of the transfer rate. This can be expected to disrupt the operation of the congestion control algorithm to some degree.



## 6. LESSONS LEARNT AND FUTURE WORK

First, and foremost, our experience has taught us that packet reordering is not innocuous, even on the scales of 0.2%. The results presented show that TFRC loss events caused by packets arriving too late and out of order can significantly affect throughput in the absence of actual packet loss.

Our implementation utilizes RTCP to provide the feedback loops needed by TFRC. Since feedback timing is important, and directly impacts calculation of the loss event rate, we are investigating the interaction between RTP and the TFRC protocol. In particular, how often loss event and round trip time information can be communicated, and how the transmission rate can be adapted.

As noted in Section 3, we piggyback feedback information into RTCP APP packets. Standard reporting intervals are on the order of seconds, too slow for effective TFRC feedback, but the reduced reporting interval of

$$T_{RTCP} = 360/B_{session} \quad (2)$$

where  $B_{session}$  is the session bandwidth expressed in kilobits per second may be used. For our application, this corresponds to a report every  $400\mu s$  on average, easily allowing feedback at least once per round trip time (although the processing load may prohibit this).

Processing load is also an issue when implementing the loss interval calculation. We noticed that our implementation observed packet loss at a lower data rate when the calculation of the TFRC parameters was enabled, even if they were not used to control the sending rate. Investigation pointed to the calculation of the average loss interval: performing this computation for every packet is a significant bottleneck, especially for high-rate sources (tests show that the loss event calculation, for a full rate HDTV source, consumes 14% of the CPU on an otherwise unloaded host).

There are also issues with rate adaptation, since the obvious method of changing the transmission rate – adapting the video frame rate – will cause significant step changes in the throughput, and cannot choose any arbitrary rate. TFRC assumes the TCP-friendly rate can be selected, and it is not clear how deviations affect the system behavior. These issues also feed into the human factors of the system: not only must the rate adaptation fit the dictates of TCP-friendly behaviour, it must be chosen to avoid disturbing viewers with sudden quality changes.

## 7. CONCLUSIONS

When discussing congestion control, it is common to focus on packet loss, since that is the primary driver in TCP, and

TCP-friendly, congestion control. There are, however, real-world IP networks in which packet loss is an extremely rare event, but where packet reordering is not infrequent. Our measurements show that this reordering limits the transmission rate of both native TCP flows, and multimedia flows controlled by the TCP friendly rate control protocol.

We understand the desire to be TCP-friendly, but it is not clear that this behavior is appropriate for multimedia applications. Indeed, one of the major philosophies in the design of RTP was Application Level Framing, making applications tolerant to packet loss and reordering. We believe that, if the network is not congested, emulation of TCP's response to packet reordering is overly conservative.

To allow the deployment of high-rate multimedia, such as HDTV-over-IP, it is necessary to develop congestion control that is both safe and usable. The TFRC protocol is clearly safe, but we have demonstrated scenarios where its overly conservative nature limits its usefulness. It is desirable to develop modifications to TFRC that decouple its response to congestion and packet reordering, so that reordering without congestion ceases to be a limiting factor.

## 8. ACKNOWLEDGMENTS

This work is supported by DARPA ITO and by hardware donated by Intel corporation.

## 9. REFERENCES

- [1] J. C. R. Bennett, C. Partridge, and N. Shectman. Packet reordering is not pathological network behavior. *IEEE/ACM Transactions on Networking*, 7(6):789–798, December 1999.
- [2] E. Blanton and M. Allman. On making TCP more robust to packet reordering. *ACM Computer Communication Review*, January 2002.
- [3] S. Floyd, M. Handley, J. Padhye, and J. Widmer. Equation based congestion control for unicast applications. In *SIGCOMM Symposium on Communications Architectures and Protocols*, 2000.
- [4] L. Gharai, G. Goncher, C. Perkins, D. Richardson, and A. Mankin. RTP payload format for SMPTE 292M. Internet Draft, Internet Engineering Task Force, February 2002. Work in progress.
- [5] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose. Modeling TCP throughput: A simple model and its empirical validation. *ACM Computer Communication Review*, 28(4):303–314, September 1998.
- [6] V. Paxson. End-to-end internet packet dynamics. *IEEE/ACM Transactions of Networking*, 7(3), June 1999.
- [7] C. S. Perkins, L. Gharai, T. Lehman, and A. Mankin. Experiments with delivery of HDTV over IP networks. In *Proceedings of the 12th International Packet Video Workshop*, Pittsburgh, April 2002.
- [8] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RTP: A transport protocol for real-time applications. IETF Audio/Video Transport Working Group, January 1996. RFC1889.

# Large Group Teleconferencing: Techniques and Considerations \*

Ladan Gharai   Colin Perkins   Allison Mankin

USC Information Sciences Institute

3811 N. Fairfax Drive, Suite 200

Arlington, VA, 22203

March 8, 2002

## Abstract

Much work has focused on the problems of small group communication and of one-to-many broadcast, while issues in large scale interactive networked teleconferences have received less attention. In this paper, we consider the problems inherent in conducting large scale conferences: teleconferences with hundreds, or perhaps thousands, of active participants. The lessons learnt from our design for a digital amphitheater – a system based on active agents, where about one hundred remote participants can conference together – are discussed. In that system we successfully overcame end system limitations by off-loading some processing into the network, thus creating parallelism and reducing the bottleneck inherent in the serial nature of the hosts managing each display. We expand on this architecture, further exploring parallelism by pushing functions from individual end systems, to clusters and the network, with the aim of scaling to thousands of users.

---

\*This paper is based upon work supported by the Information Technology Office of the Defense Advanced Research Projects Agency. Any opinions, findings, conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of DARPA. The authors may be contacted as {ladan|csp|mankin}@isi.edu

# 1 Introduction

The infrastructure of the Internet has grown at an incredible pace since its inception. With the widespread introduction of optical networking, bandwidth has become plentiful and wide-area networks operating at OC-48 rates, such as the SuperNet <sup>1</sup> and Internet2 <sup>2</sup>, are common with OC-192 and other higher rate connections becoming available. This growth in wide area connectivity has been matched by improvements in LAN technology: 100 Mbit Ethernet is ubiquitous, 1 gigabit is becoming common, and 10 gigabit Ethernet is now being introduced.

Parallel to the growth of the infrastructure, consumers and academics have been busy envisioning new and far reaching applications: the traditional uses – email, netnews, file transfer – are giving way to more interactive applications based on the world wide web, to streaming media, digital radio, television and cinema, and to real-time interactive teleconferencing.

There is another variable which affects this happy growth in network bandwidth and application demand: the performance of the end-systems. According to Moore's Law, the number of transistors that will fit on a chip doubles every 18 months, and performance closely follows this. This is an impressive increase, but is dwarfed by the rate of increase in network capacity, which has grown at a much faster pace (figure 1). Given the availability OC-48 PCI network interface cards, gigabit – and soon 10 gigabit – Ethernet cards, the question remains: is current processing power capable of processing such data rates?

In this paper we explore end-system limitations in the context of scaling video conferencing, tele-conferencing hundreds, or perhaps thousands, of participants. Such a conference may be held worldwide with participants from different university campuses, corporate facilities, government organizations or even a lone participant from the Arctic. In this work we draw from our experience with the digital amphitheater, where we successfully video conferenced close to one hundred participants.

We begin, in section 2, by further describing the architectural implications of scaling teleconferencing systems.

---

<sup>1</sup><http://www.ngi-supernet.org/>

<sup>2</sup><http://www.internet2.org/>

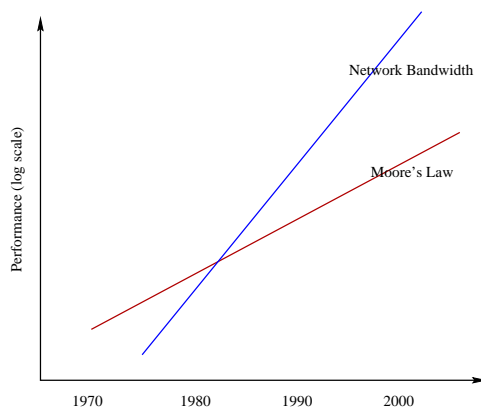


Figure 1: Moore's law vs. network growth.

Next, in section 3, we briefly describe our initial experiences with a prototype large-scale conferencing system – the digital amphitheater – and the lessons learnt from it. We expand on this experience to explore issues related to scaling up further, to possibly thousands of participants, in section 4. Finally we discuss related work in this area, section 5, and conclude in section 6.

## 2 Architectural Implications of Scaling

We envision a system capable of supporting several hundred, perhaps one thousand, simultaneous interactive users. The benefits of such a system are obvious: large organizations can have regular meetings with all levels of management involved without incurring high travel cost, long distance educational programs can meet as if within a lecture hall while students and lecturers join from geographically disparate locations, or it could be used for political and other debates.

Video teleconferencing among small groups of people is now quite common, and is supported by a number of commercial and open-source tools. However large structured meetings, on the scale that we are envisioning, have not yet been tried. There are a number of reasons for this: processing such a large number of video streams

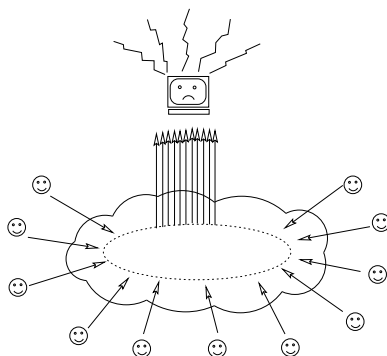


Figure 2: The end-system bottleneck.

presents a formidable challenge, both in the network and for the end-user application, and display technology is often a limiting factor. Processing over a thousand video streams can easily overwhelm most consumer grade workstations, in terms of bus access, interrupt processing, context switching, packet handling and demultiplexing, decoding, display processing and rendering.

Many of the current teleconferencing tools, especially the research oriented ones such as the popular multicast toolset maintained by University College London [13] have been designed with scaling properties in mind. However, their focus has been mainly on attaining scaling via multicast, and thereby reducing network load. This approach does not address the problem of the end-system bottleneck, and in fact it aggravates it. End-users can generate video content in parallel, this content moves through the network, but once received at its destination, must be processed by an inherently serial system. As all the video flows must be instantaneously reconstructed, decompressed and rendered (figure 2), thereby creating a performance bottleneck in the end-system.

Given that the processing limitations of end-systems are the main bottleneck and deterrent to very large scale video conferencing, what are the possible solutions? Our experience shows that the simple brute force technique of ‘faster end-systems’ is not a viable solution, as even the fastest available workstations cannot keep up with hundreds of video streams.

The implication is that we must distribute the processing, leveraging the increased communication ability rather

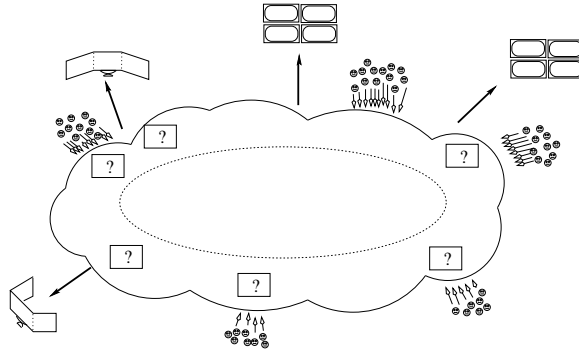


Figure 3: Large group conferencing: An architectural template.

than drinking from the firehose of the full set of input streams. Parts of processing must be pushed into the network infrastructure, offloading functions from the end-system to agents within the network (figure 3). The questions remains as to how much and which parts of the process can be off-loaded from the end-system, and exactly what are the tradeoffs involved.

We have explored some of these tradeoffs, and the performance which can be gained through the use of agents, in our prototype digital amphitheater. In the next section we discuss this in some detail, followed by an evaluation of the extensibility of this model to other large scale conferences.

### 3 The Digital Amphitheater: Prototyping Large-Scale Conferencing

When attempting to conduct a teleconference with upwards of one hundred participants using the standard multi-cast toolset, it rapidly becomes clear that those tools are unable to process the received data: the system load goes to 100%, data is dropped, and the visual appearance of the conference is destroyed.

There can be a number of reasons for this poor performance: it could be that the system cannot handle the total bandwidth of the incoming media streams, it could be the per-packet processing, it could be limited CPU performance, or it could be limited memory bandwidth in the host.

Our initial investigation led us to believe that the limit was not related to the raw bandwidth of the media streams: experiments with high rate TCP and UDP traffic on similar hosts have shown that they can receive significantly higher data rates, if tuned correctly. Those experiments also suggest a number of approaches to tuning the system, including: use of large frames, interrupt coalescing, zero copy networking, and checksum offloading [3].

When considering these, our first observation was that the media streams comprise a large number of small packets, due to the compressed nature of their payload. If these packets could be combined into larger frames, it might be possible to increase performance without having to tune the end host. This can also be expected to ease the application performance, by reducing the number of participants it must track.

This is a simple matter for TCP streams, since the communication is point to point and the data can be split arbitrarily. For real-time communication, however, the problem is more complex due to the following factors: (1) a video conferencing session naturally involves multiple sources; (2) the size of the packets generated depends on the compression scheme used, and cannot be arbitrarily varied. In particular, any change in the compression ratio to affect the packet size will vary the rate, defeating the point of the change.

We devised a technique we have term ‘Spatial Tiling’ which address the above constraints: combining data from multiple sources whilst maintaining the rate [7].

### **3.1 Spatial Tiling**

Our concept of spatial tiling is to tile  $N$  frames from separate sources next to each other, and to modify the meta-data of the tiled frame, such that it represents a single frame. This is illustrated via an example in figure 4 where three individual video frames are placed side by side to form a single frame. Each individual frame is completely represented in the tiled frame, however the meta-data, in this case block coordinates, has been adjusted accordingly. We believe tiling satisfies both of our constraints: packetizing the tiled frame provides more opportunity for generating fuller packets, and the number of input sources is reduces from  $N$  to a single source.

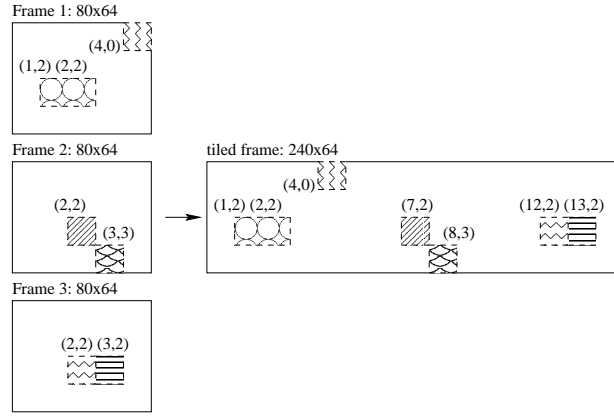


Figure 4: Tiling three frames into a single frame. Both frame size and block coordinates have been adjusted for the tiled frame.

It is important that spatial tiling does not add additional delay to the video stream. Tiling agents only parse and deconstruct the incoming video streams into smaller building blocks, whilst maintaining their relevant meta-data: no decompression is done in the tiling agent. To maintain independence between incoming and outgoing frame-rates, two sets of buffers are maintained per stream. The tiled frame is constructed at given intervals (determined by the outgoing frame rate) from the output buffers. New incoming frames are copied from the incoming buffer to the output buffers, once they are received in full.

Although, theoretically, it is possible to tile an unlimited number of streams, we have restricted the tiling to 15 video streams. This restriction allows us to use the built in mixer functionality of RTP/RTCP [15], since an RTP packet can carry the contributing source identifiers for up to 15 different sources. The input streams can be tiled in any geometry requested: for 15 streams the agent can generate a single row of 15x1, a square of 4x4 (where the last square will be empty), a 5x3 rectangle, or even a single row/column.

In our current implementation, the spatial tiling agents support two video representations: high bandwidth raw YUV video with conditional replenishment (YUVCR) [8] and H.261 [17] using only intra-frame compression. Spatial tiling agents may be employed within a standard video conferencing session, or in conjunction with a



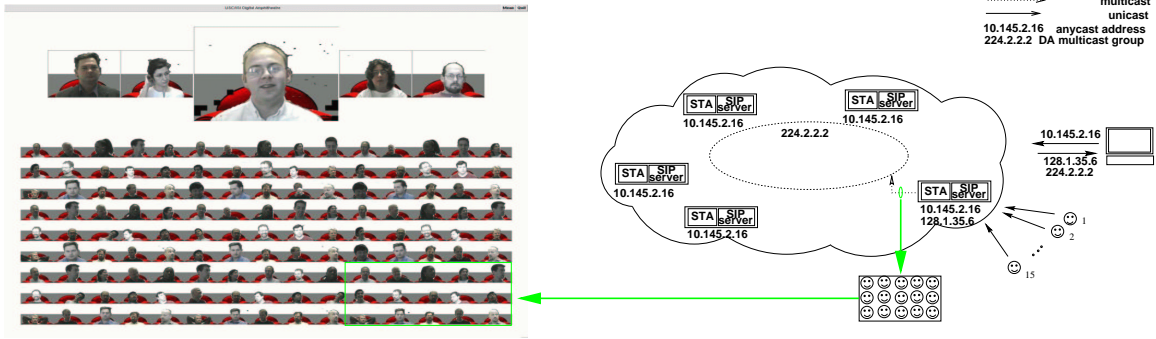


Figure 5: The digital amphitheater: user interface and architecture.

special purpose application, such as our digital amphitheater [14] system.

Figure 5 displays the digital amphitheater’s user interface (with a conference in session). The amphitheater consists of the rows of attendees, four front panel members and the speaker. In all instances the background of the offices have been removed, in order engender a feeling of presence and location. To do so we have adapted a simple, low cost background substitution algorithm which runs in real time on the senders systems.

Each attendee in the amphitheater participates by unicasting video to the ‘closest’ agent, located via an anycast address. The agent, in turn, tiles together all the video streams it receives, and multicasts the tiled video stream to a multicast group. All participants join this group, receiving and displaying the combined audience video. The panel members and speaker send directly to the multicast group, thus avoiding the tiling.

### 3.2 Performance Gains

To determine the performance gains obtained by using the tiling agents we decided to measure and quantify: (1) bandwidth, in bits per second (*bps*); (2) packets per second (*pps*); and (3) the total number of streams the end system is capable of decoding and rendering (*N*). We compared the value of these variables in a conferencing session with and without the use of STAs. Here, we only present results obtained from the H.261 tests (results

Variable	seperate	tiled	gain%
<i>pps</i>	186	122	34.51%
<i>bps</i>	979	936	4.33%
<i>N</i>	55	97	43.2%

Table 1: Variables quantified with and without the STAs for H.261 video: average bit-rate, *bps*, average packet rate, *pps*, and *N* total number of streams.

with YUVCR support these conclusions, and are omitted due to lack of space).

The receiving system was what is currently considered an average user grade system: a 550Mhz Pentium III machine with 256M of memory, running Red Hat Linux 7. The tiling was initially run on a somewhat lower grade system, a 400Mhz Pentium II with 64M memory, running FreeBSD 3.4, but this was found to have insufficient memory, although it was sufficient in other ways. A more powerful system, with 512M of memory, was used to host the tiling agents during the tests we report. Work is underway to reduce the memory footprint of the tiling agents, since they are otherwise not very compute intensive and require only a few percentage of CPU time.

In our initial set of trials, we measured *bps* and *pps*. To do so, first we streamed the 15 test video streams individually to the receiver. Next, we ran the test video through the tiling process with the output frame rate set to 8 fps - essentially the same as the input frame rate of the test videos. To measure the bitrate, *bps*, and packet rate, *pps*, we instrumented the receiver such that it logged these variables, along with other decoding statistics, to a file. Figure 6 displays the results of these tests. In these graphs, *pps* clearly show a reduction for the tiled H.261 stream. Although not apparent, *bps*, also shows an overall reduction of 4.33% percent for the tiled stream over the entire test run. Table 1 summarizes these results.

The reduction in *pps* is primarily due to the aggregation of smaller packets. The tiling process generates a single large frame, therefore there are fewer ‘half empty’ packets in the resulting stream. In the tiled H.261 stream *pps* is reduced by approximately 35%. Given the average size of H.261 packets such a decrease is to be expected.

Figure 7 displays the cumulative packet size for 15 individual H.261 streams and their corresponding tiled video frame.

In terms of bandwidth, *bps* is reduced by 4%. Although bandwidth is reduced over the duration of the test runs, the graphs reveal that this is not the case on a per minute bases, as in some instances the *bps* of the separate streams appears to be less than the tiled stream. This is in part due to synchronization differences between the separate streams and the tiled stream, and in part due to measurement artifacts resulting from the averaging process. We also note the the low reduction in bandwidth is to be expected. In these tests the tiling agents reproduce the input video streams, exactly as they come in, without any temporal or spatial down sampling. Both the input and output frame rates are 8 fps and the tiling agents more or less copy each incoming frame to the outgoing tile frame. The existing reduction in *bps* is mainly a reflection of the reduced *pps* and lower packet overhead,

Finally, we turned our attention to the performance of the end-system, and quantifying  $N$ . Our decoder maintains statistics on the number of packets correctly decoded and on packets discarded due to late arrival or lack of rendering time. We used these statistics to measure the maximum number of streams,  $N$ , our end-system could receive without loss, both with and without tiling. This process was conducted by incrementally increasing the number of individual streams until the end-system reached the point of saturation. For the H.261 video streams it was found that the system could decode and render up to 55 individual video streams without loss. With this number of streams CPU was at 100% utilization. When receiving tiled H.261, the system could receive 6 tiled streams of 15 and an additional stream of 2 tiles, comprising a total of 97 individual streams, an overall increase of 43% in number of streams.

These numbers clearly demonstrate the reduction of workload on the end-system due to the spatial tiling process. Despite almost no reduction in bit-rate, the end-system is capable of receiving almost twice as many video streams, once the video streams are tiled and packet rate is reduced. This leads us to conclude that a primary load on end-systems is per-packet interrupt processing and per-source rendering, rather than the computational complexity of the decoding process and therefore spatial tiling is more amendable to relatively highly compressed video streams

where the average packet size is significantly smaller than the network MTU. Having a significant number of ‘half-full’ packets, gives the tiling agents more leverage in reducing the overall packet rate.

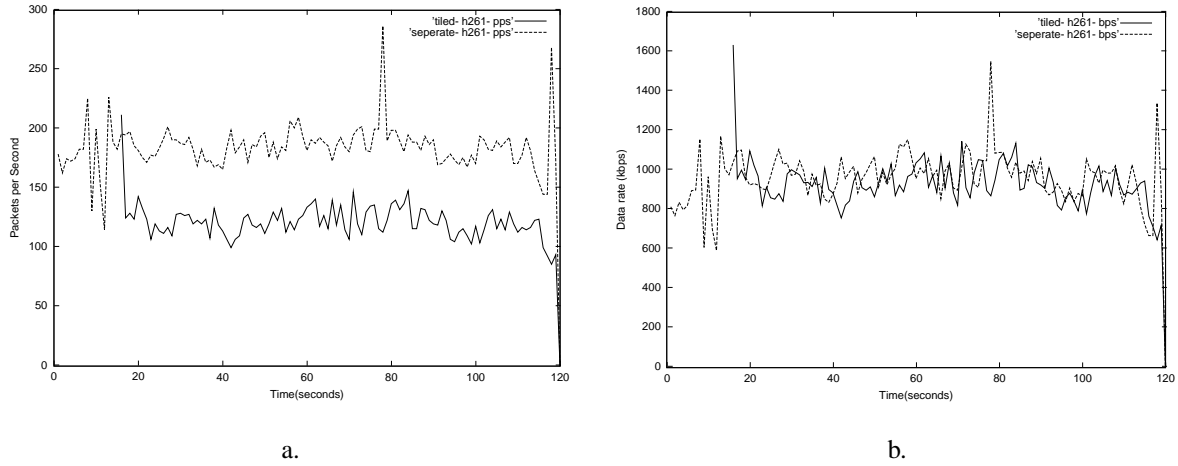


Figure 6: Comparison of 15 separate H261 QCIF video streams and the equivalent 5x3 tiled stream (a) packets per second (b) bits per second.

### 3.3 Limitations and Lessons Learnt

The spatial tiling agents were designed with two goals in mind: (1) reducing the number of sources visible to the receiver; and (2) reducing the number of smaller packets, combining them into larger output. Clearly this has resulted in performance increases: the number of streams received successfully by our end-system has almost been doubled, whereas the number of streams has actually been reduced from 55 to 5, as the 97 tiled streams, are really 5 separate streams.

By allowing the tiling agents to packetize bigger frames, we were able to produce 35% less packets of which over 80% are at MTU size (figure 7). This is in contrast to having only 40% of packets at full capacity.

Although, it is very promising that 80% of the packets are mostly full, it also indicates that we have reached the limit of what can be achieved via spatial tiling. Employing a hierarchy of tiling agents, or tiling more than 15

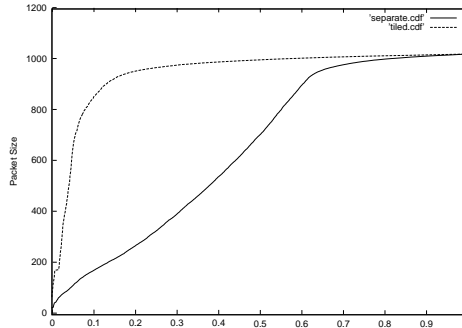


Figure 7: Cumulative distribution of packet sizes, showing increase in the fraction of large packets after tiling

streams, may reduce the number of sources received by an end-system, but it is unlikely that it reduce *pps* any further.

## 4 Challenges in Further Scaling

Experience with the digital amphitheater has shown the benefit which can be gained by coalescing packets within the network to reduce the load on an end-system. This is effective at reducing the packet rate up to a point, but rapidly runs into a bottleneck due to limitations of the network MTU.

To further scale the system we need to consider the other limiting factors, starting with general issues in optimizing the network stack, and moving on to issues with efficient RTP processing.

### 4.1 Optimizing the network stack

There are a number of ways in which the performance of the network stack can be improved, many of which have been explored in the context of high performance TCP implementations. These optimizations fall into two major categories: those which improve the per-packet performance of the stack, and those which reduce the per-byte overheads.

The major issue with respect to per-packet overheads is the interrupt processing load. High performance TCP implementations reduce this by sending larger packets, a solution which we have explored for teleconferencing through our use of spatial tiling.

Another solution is interrupt coalescing, where the network driver gathers several packets before signaling to the operating system that data has been received, rather than generating an interrupt for each packet. This is clearly of benefit, and needs little further discussion.

Reduction in the per-byte overhead is typically achieved with zero copy network stacks, where the network hardware writes into a buffer which is directly mapped into the application's memory space. This is useful, up to a point, but the gains which can be achieved are perhaps limited compared to some other applications.

The main issue is decompression of the media stream, which not only results in a copy of the data being made, but it's expansion. For example, a factor of ten compression is not unreasonable, yet the memory traffic generated by such a decompressor will dwarf the gains from a zero-copy network stack.

The other common technique used to improve performance of TCP is checksum offloading, where the network interface verifies the TCP or UDP checksum before passing the packet to the host. Once again, there is some gain for teleconferencing applications, but due to the processor intensive nature of these applications it may have limited impact (for example, the cost of computing a UDP checksum is negligible compared to that of MPEG decompression).

To conclude, there is some gain to be had from optimizing the UDP/IP network stack, but the nature of large scale teleconferencing is such that the RTP and application processing costs dwarf those of UDP/IP network processing. The reduction in packet rate achieved through spatial tiling is perhaps the most significant effect we can expect, followed by the gains from zero-copy stacks.

## 4.2 Optimizing RTP

Rather than considering the network stack alone, it is instructive to view the complete application, and consider how performance of the RTP protocol and media decoding can be optimized. Such an approach fits well with the nature of RTP, which was designed around the concepts of application level framing [4] and integrated layer processing.

There are two major issues to consider when optimizing RTP and application performance: participant state and media decoding performance.

An application using RTP will maintain a significant amount of state for each participant: the 32 bit synchronization source identifier, playout calculation details, decoding state, media data awaiting playout, reception quality statistics, source description information, etc. In total, this can easily comprise several hundred bytes per participant, excluding the media data.

When compared to the cache sizes of modern processors, where 64kBytes is considered large, it is clear that the complete state cannot fit into the level 1 cache. It may therefore be advantageous to split data structures into those parts necessary for decoding each packet and those parts which are required less often, so that unnecessary references to main memory can be avoided.

The use of tiling agents within the network has the unintended consequence of reducing the amount of state which has to be accessed during decoding. Since they act as synchronization sources for the media, it is necessary for the receiver to maintain state per-agent, rather than per participant.

As we have noted, media decoding is significant in terms of both processor utilization and memory bandwidth. The obvious result of this is that zero copy techniques must be adopted within an application, with media being rendered directly onto the display device if possible. In addition, the use of various SIMD extensions to processor instruction sets (e.g. MMX, AltiVec, VIS, etc.) can significantly reduce decoding times.

Media decoding is also a function which is parallelizable: each member of the session can be independently

decoded and rendered. There is a clear benefit to be gained through the use of multiprocessor hardware, but this still suffers from a bottleneck due to the single network socket used to receive data.

We believe that there is a significant amount to be gained through the use of layered coding, in conjunction with multi-processor systems. The use of layered coding allows the network interface card to filter unwanted traffic, so each processor sees only a fraction of the total. This gives the benefit of parallel decoding, along with a signification reduction in the amount of state which needs to be kept.

It may be advantageous for agents near the edges of the network to combine outgoing data into a single stream, and split incoming data into layers. The trade-off for optimal performance at the end host and in routers is different: it is better for data to be layered at the edges, so the host can separate processing, but it is better to be combined in the core where packet switching is not an issue but per group state maintenance is. This is an ideal use for agents: offloading processing from groups of hosts at the boundary between relatively low speed local networks, and the high speed core network.

## **5 Related Work**

The use of active service agents [1] to adapt the behavior of network traffic flows has been widely studied. Active services avoid the well known problems of active networks by restricting computations to the application layer, deploying services onto a network of computation servers placed within the network. As a result of this, they are readily deployable and form the basis of a number of commercial content distribution networks.

Whilst these commercial offerings have typically focused on efficient distribution of world-wide web content, a number of researchers have studied the problem of adapting streaming audio/video flows to match the network capacity.

One of the earliest such papers referred to self-organized transcoding of streaming audio/video media flows [11], leveraging from tools such as the video gateway developed at UC Berkeley [2]. More recently, implementations



such as the ‘FunnelWeb’ Application Level Active Network [6], active routers [10], and overlay networks, such as the X-bone [16], add genericity and flexibility to the system.

The use of these techniques, whilst beneficial to the network, degrades the quality of the media stream. It would be desirable if the load generated by a media stream could be reduced whilst retaining its quality. Our proposed STA network has this property, at the expense of limited adaptability (when compared to schemes based on transcoding).

Critical to the operation of active services is the placement of the active elements within the network. This has received considerable attention in the literature [18, 12, 5], particularly when related to reliable multicast, leading to recent standards work in the IETF [9]. We do not seek to design new tree building mechanisms at present, rather we rely on existing work.

Our proposal seeks to leverage existing work in the field of active services: the concept of active agents within the network to adapt media flows to fit network/system constraints, the platforms for service creation and deployment, and mechanisms for placement of service agents.

## 6 Conclusion

We have presented the digital amphitheater, a system for large-scale teleconferences, based around the use of active agents to tile video streams, reducing the load on the receivers. This system illustrates one approach to scaling a teleconferencing system to large numbers of participants.

We have also present some preliminary thoughts on how we can further scale the system, to larger or higher quality conferences. In particular, we note the beneficial effect of RTP mixer/translators in reducing the state requirements for end systems, and their potential role layering media streams to enable efficient parallel decoding.

The requirements for conducting large conferences are difficult to meet, and no existing application is entirely

successful. The digital amphitheater is a step towards the solution, and points the way to further development and performance improvements.

## References

- [1] E. Amir, S. McCanne, and R. Katz. An active service framework and its application to real-time multimedia transcoding. In *Proc. ACM SIGCOMM 1998*, Vancouver, BC, 1998.
- [2] E. Amir, S. McCanne, and H. Zhang. An application level video gateway. In *Proc. ACM Multimedia '95*, San Francisco, CA, November 1995.
- [3] J. Chase, A. J. Gallatin, and K. G. Yocum. End system optimizations for high-speed tcp. *IEEE Communications Magazine*, 39(4):68–74, April 2001.
- [4] D. D. Clark and D. L. Tennenhouse. Architectural considerations for a new generation of protocols. *Computer Communications Review*, 20(4), September 1990.
- [5] P. Francis. Yallcast: Extending the Internet multicast architecture. Technical report, NTT Information Sharing Platform Laboratories, September 1999.
- [6] M. Fry and A. Ghosh. Application level active networking. *Computer Networks*, 31(7):655–667, July 1999.
- [7] L. Gharai, C. Perkins, and A. Mankin. Scaling video conferencing through spatial tiling. In *Proc. NOSSDAV 2001*, Port Jefferson, NY, June 2001.
- [8] M. Handley. YUV-CR codec for vic. Personal correspondence.
- [9] M. Kadansky, B. Levine, D. M. Chiu, B. Whetten, G. Taskale, B. Cain, D. Thaler, and W. H. Koh. Reliable multicast transport building block: Tree auto-configuration. Internet Engineering Task Force, November 2000. Work in progress.

- [10] R. Keller, S. Choi, D. Decasper, and M. Dasen. An active router architecture for multicast video distribution. In *Proc. IEEE Infocom 2000*, Tel Aviv, March 2000.
- [11] I. Kouvelas, V. Hardman, and J. Crowcroft. Network adaptive continuous-media applications through self organised transcoding. In *Proc. NOSSDAV '98*, Cambridge, UK, July 1998.
- [12] B. N. Levine, S. Paul, and J. J. Garcia-Luna-Aceves. Organizing multicast receivers deterministically by packet loss correlation. In *Proc. ACM Multimedia '98*, Bristol, UK, September 1998.
- [13] University College London. Multicast conferencing applications archive. Software available online, 2001. <http://www-mice.cs.ucl.ac.uk/multimedia/software/>.
- [14] A. Mankin, L. Gharai, R. Riley, M. Perez Maher, and J. Flidr. The design of a digital amphitheater. In *Proc. NOSSDAV 2000*, Chapel Hill, NC, June 2000.
- [15] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RTP: A Transport Protocol for Real-time Applications. Internet Engineering Task Force, January 1996. RFC 1889.
- [16] J. Touch and S. Hotz. The X-Bone. In *Proc. 3rd Global Internet Mini-Conference/Globecom '98*, Sydney, November 1998.
- [17] International Telecommunication Union. Video codec for audiovisual services at Px64 kbits/s. ITU-T recommendation H.261, 1993.
- [18] R. X. Xu, A. C. Myers, H. Zhang, and R. Yavatkar. Resilient multicast support for continuous media applications. In *Proc. NOSSDAV '97*, Washington University in St. Louis, May 1997.

# Yima: A Second-Generation Continuous Media Server

**Yima, a scalable real-time streaming architecture, incorporates lessons learned from earlier research prototypes to enable advanced continuous media services.**

Cyrus  
Shahabi

Roger  
Zimmermann

Kun Fu

Shu-Yuen  
Didi Yao

University of  
Southern California

Applications such as news on demand, distance learning, e-commerce, and scientific visualization all store, maintain, and retrieve large volumes of real-time data over a network. These data are denoted collectively as *continuous media*, or CM. Video and audio objects are popular examples; haptic and avatar data are less familiar types. CM data require a streaming architecture that can, first, manage real-time delivery constraints. Failure to meet these constraints on CM data disrupts the display with “hiccups.” Second, the architecture must address the large size of CM objects. A two-hour MPEG-2 video with a bandwidth requirement of 4 megabits per second is 3.6 gigabytes in size.

The currently available commercial implementations of CM servers fall into two broad categories:

- low-cost, single-node, consumer-oriented systems serving a limited number of users; and
- multinode, carrier-class systems such as high-end broadcasting and dedicated video-on-demand systems.

RealNetworks, Apple Computer, and Microsoft product offerings fit into the consumer-oriented category, while SeaChange and nCube offer solutions oriented toward carrier-class systems. While commercial systems ordinarily use proprietary technology and algorithms, making it difficult to compare their products with research prototypes, we have designed and developed a second-generation CM server that demonstrates several advanced concepts.

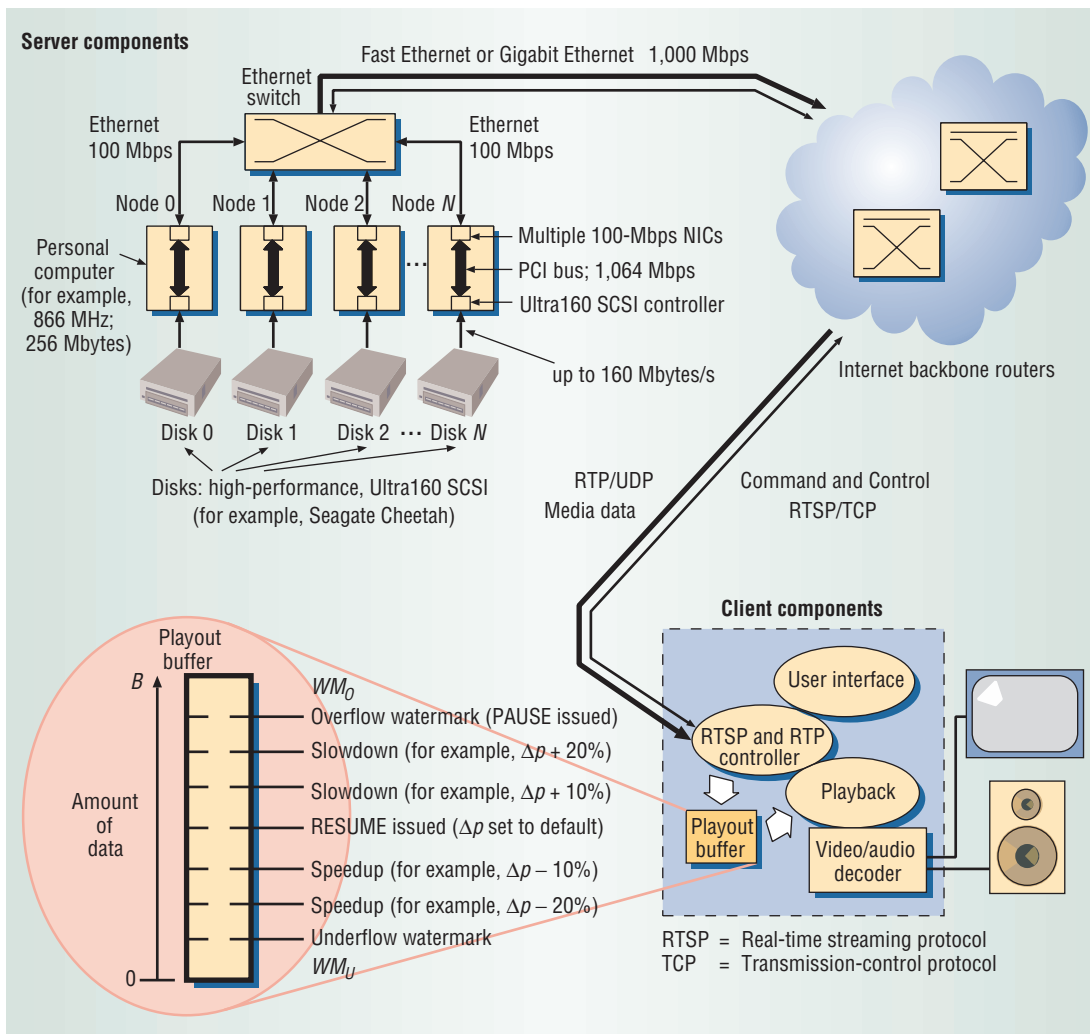
We call our system Yima, a name denoting the first man in ancient Iranian religion. While Yima has not achieved the refinement of commercial solutions, it is operational and incorporates lessons learned from first-generation research prototypes.<sup>1,2</sup> Yima distinguishes itself from other similar research efforts in the following:

- complete distribution with all nodes running identical software and no single points of failure;
- efficient online scalability allowing disks to be added or removed without interrupting CM streams;
- synchronization of several streams of audio, video, or both within one frame (1/30 second);
- independence from media types;
- compliance with industry standards;
- selective retransmission protocol; and
- multithreshold buffering flow-control mechanism to support variable bit-rate (VBR) media.

Yima is also a complete end-to-end system that uses an IP network with several supportable client types. This feature distinguishes it from previous research that focused heavily on server design.

## SYSTEM ARCHITECTURE

Figure 1 shows the overall Yima system architecture. In our prototype implementation, the server consists of an eight-way cluster of rack-mountable Dell PowerEdge 1550 Pentium III 866-MHz PCs with 256 Mbytes of memory running Red Hat



**Figure 1. Yima system architecture.** The prototype implementation uses off-the-shelf commodity hardware components and industry standards end to end.

Linux. Sixteen 36-Gbyte Seagate Cheetah hard-disk drives store the media data and connect to the server nodes via Ultra160 small computer system interface (SCSI) channels.

The nodes in the cluster communicate with each other and send the media data via multiple 100-Mbps Fast Ethernet connections. Each server is attached to a local Cabletron 6000 switch with either one or two Fast Ethernet lines. The local switch connects to both a WAN backbone for serving distant clients and a LAN environment for local clients. Our testbed also includes server clusters at other remote locations, for example, Metromedia Fiber Network in El Segundo, California, and Information Sciences Institute East in Arlington, Virginia.

Choosing an IP-based network keeps the per-port equipment cost low and makes the system immediately compatible with the public Internet.

The current prototype implements clients on standard Pentium III PC platforms, but we could also port them to digital television set-top boxes. The client software, Yima Presentation Player, runs on either Red Hat Linux or Windows NT. Structured into several components, the player lets various software and hardware decoders be plugged in. Table 1

shows the different media types that Yima currently recognizes. One unusual type is panoramic video with 10.2-channel audio.

## SERVER DESIGN CHALLENGES

The servers for delivering isochronous multimedia over IP networks must store the data efficiently and schedule the data retrieval and delivery precisely before transmission. We studied both master-slave and bipartite design approaches in the Yima-1 and Yima-2 CM servers, respectively. These approaches share many features that address design challenges in this domain. They differ mainly in the logical interconnection topology between cluster nodes.

## Data placement and scheduling

There are two ways to assign data blocks to the magnetic disk drives that form the storage system: in a round-robin placement<sup>3</sup> or randomly.<sup>4</sup> Traditionally, round-robin placement uses a cycle-based approach for resource scheduling to guarantee a continuous display, while random placement uses a deadline-driven approach.

In general, the round-robin cycle-based approach provides high throughput with little wasted band-

**Table 1. Yima client media support.**

Media type	Decoder	Channels	Operating system	Minimum CPU speed	Video resolution (in pixels)	Audio encoding	Delivery rate
DivX MPEG-4	Software	1 video, 2 audio	Linux	500 MHz	720 × 480	MP3	<1 Mbps
MPEG-2 and Dolby Digital	Creative Dxr2 DVD	1 video, 5.1 audio	Linux	200 MHz	720 × 480	Dolby AC-3	6-8 Mbps
MPEG-2 HD	Software	1 video	Linux	>2 × 1.5 GHz	1,920 × 1,080		19.4 Mbps
MPEG-2 HD	Vela Research CineCast HD	1 video, 10.2 audio	Linux	500 MHz	1,920 × 1,080	Dolby AC-3 or uncompressed PCM	19.4-45 Mbps and 11 Mbps
Panoramic MPEG-2	Vela Research CineCast	5 video, 10.2 audio	Windows NT	2 × 400 MHz	(5 × 720) × 480 each	Uncompressed PCM	4 × 5 Mbps and 11 Mbps

width for video objects that are retrieved sequentially, such as a feature-length movie. The startup latency for an object might be large under heavy loads, but object replication can reduce it.<sup>5</sup>

The random deadline-driven approach supports fewer optimizations, so it could lower throughput, but several benefits outweigh this potential drawback.<sup>6</sup> First, random data placement supports multiple delivery rates with a single server block size; it also simplifies the scheduler design, supports interactive applications, and automatically achieves the average transfer rate with multizoned disks. Finally, random placement reorganizes data more efficiently when the system scales up or down.

Random placement can require a large amount of metadata to store and manage each block's location in a centralized repository, for example, in tuples of the form  $\langle \text{node}_x, \text{disk}_y \rangle$ . Yima avoids this overhead by using a pseudorandom block placement. A seed value initiates a sequence of numbers that can be reproduced by using the same seed value. By placing blocks in a pseudorandom fashion across the disks, the system can recompute the block locations. Since Yima numbers disks globally across the server nodes, it will assign blocks to random disks across different nodes.

Hence, Yima stores only the seed for each file object instead of locations for every block.

### Scalability, heterogeneity, and fault resilience

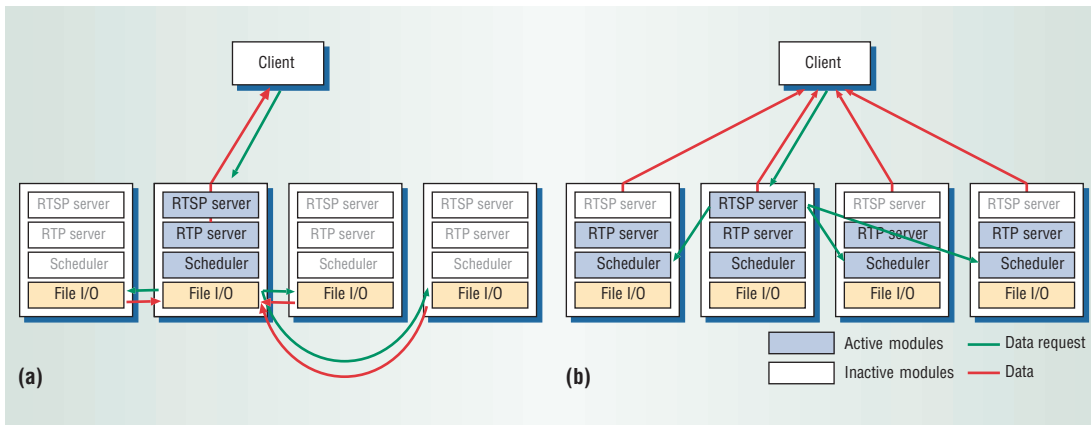
Any CM server design must scale to support growth in user demand or application requirements. Several techniques address this requirement, including the use of multidisk arrays. However, if the design connected all the disks to a single large computer, the I/O bandwidth constraints would limit the overall achievable throughput—hence, Yima's architecture uses multiple computers, or multinodes.

As Figure 1 shows, the Yima server architecture interconnects storage nodes via a high-speed network fabric that can expand as demand increases. This modular architecture makes it easy to upgrade older PCs and add new nodes.

Applications that rely on large-scale CM servers, such as video-on-demand, require continuous operation. To achieve high reliability and availability for all data stored in the server, Yima uses disk merging<sup>7</sup> to implement a parity-based data-redundancy scheme that, in addition to providing fault tolerance, can also take advantage of a heterogeneous storage subsystem. Disk merging presents a virtual view of logical disks on top of the actual physical storage system, which might consist of disks that provide different bandwidths and storage space. This abstraction allows a system's application layers to assume a uniform characteristic for all the logical disks, which in turn allows using conventional scheduling and data placement algorithms across the physical storage system.

### Data reorganization

Computer clusters try to balance load distribution across all nodes. Over time, both round-robin and random data-placement techniques distribute data retrievals evenly across all disk drives. When a system operator adds a node or disk, however, the system must redistribute the data to avoid partitioning the server. Reorganizing the blocks involves much less overhead when the system uses random rather than round-robin placement. For example, with round-robin striping, adding or removing a disk requires the relocation of almost all data blocks. Randomized placement requires moving only a fraction of the blocks from each disk to the added disk—just enough to ensure that the blocks are still randomly placed to preserve the load balance.



**Figure 2. Client session view of the Yima server. (a) Data is sent through the master node in Yima-1, and (b) data is sent from all the nodes in Yima-2.**

Yima uses a pseudorandom number generator to produce a random, yet reproducible, number sequence to determine block locations. Because some blocks must move to the added disks when the system scales up, Yima cannot use the previous pseudorandom number sequence to find the blocks; therefore, Yima must derive a new random number sequence. We use a composition of random functions to determine this new sequence. Our approach—termed Scaling Disks for Data Arranged Randomly (Scaddar)—preserves the sequence’s pseudorandom properties, resulting in minimal block movements and little overhead in the computation of new locations.<sup>8</sup> The Scaddar algorithm can support disk scaling while Yima is online.

### Multinode server architecture

We built the Yima servers from clusters of server PCs called nodes. A distributed file system provides a complete view of all the data on every node without requiring individual data blocks to be replicated, except as required for fault tolerance.<sup>7</sup> A Yima cluster can run in either a master-slave or bipartite mode.

**Master-slave design (Yima-1).** With this design, an application running on a specific node operates on all local and remote files. Operations on remote files require network access to the corresponding node. The Yima-1 software consists of two components:

- the Yima-1 high-performance distributed file system, and
- the Yima-1 media streaming server.

As Figure 2a shows, the distributed file system consists of multiple file I/O modules located on each node. The media-streaming server itself is composed of a scheduler, a real-time streaming protocol (RTSP) module, and a real-time protocol (RTP) module. Each Yima-1 node runs the distributed file system, while certain nodes also run the Yima-1 media-streaming server. A node running only the file I/O module has only slave capabilities, while a node that runs both components has master and slave capabilities.

A master server node is a client’s point of contact during a session. We define a session as a complete RTSP transaction for a CM stream. When a client wants to request a data stream using RTSP, it connects to a master server node, which in turn brokers the request to the slave nodes. If multiple master nodes exist in the cluster, this assignment is decided based on a round-robin domain name service (RR-DNS) or a load-balancing switch. A pseudorandom number generator manages the locations of all data blocks.

Using a distributed file system obviates the need for applications to be aware of the storage system’s distributed nature. Even applications designed for a single node can to some degree take advantage of this cluster organization. The Yima-1 media streaming server component, based on Apple’s Darwin Streaming Server (DSS) project (<http://www.open-source.apple.com/projects/streaming/>), assumes that all media data reside in a single local directory. Enhanced with our distributed file system, multiple copies of the DSS code—each copy running on its own master node—can share the same media data. This also simplifies our client design since it sends all RTSP control commands to only one server node.

Finally, Yima-1 uses a pause-resume flow-control technique to deliver VBR media. A stream is sent at a rate of either  $R_N$  or zero megabits per second, where  $R_N$  is an estimated peak transfer rate for the movie. The client issues pause-and-resume commands to the server depending on how full the client buffer is. Although the pause-resume design is simple and effective, its on-off nature can lead to bursty traffic.

With the Yima-1 architecture, several major performance problems offset the ease of using clustered storage, such as a single point of failure at the master node and heavy internode traffic. These drawbacks motivated the design of Yima-2, which provides a higher performing and more scalable solution for managing internode traffic.

**Bipartite design (Yima-2).** We based Yima-2’s bipartite model on two groups of nodes: a server group and a client group.



With Yima-1, the scheduler, RTSP, and RTP server modules are all centralized on a single master node from the viewpoint of a single client. Yima-2 expands on the decentralization by keeping only the RTSP module centralized—again from the viewpoint of a single client—and parallelizing the scheduling and RTP functions, as Figure 2b shows. In Yima-2, every node retrieves, schedules, and sends its own local data blocks directly to the requesting client, thereby eliminating Yima-1's master-node bottleneck. These improvements significantly reduce internode traffic.

Although the bipartite design offers clear advantages, its realization imposes several new challenges. First, clients must handle receiving data from multiple nodes. Second, we replaced the original DSS code component with a distributed scheduler and RTP server to achieve Yima-2's decentralized architecture. Last, Yima-2 requires a flow-control mechanism to prevent client buffer overflow or starvation.

With Yima-2, each client maintains contact with one RTSP module throughout a session for control information. For load-balancing purposes, each server node can run an RTSP module, and the decision of which RTSP server to contact remains the same as in Yima-1: RR-DNS or switch. However, contrary to the Yima-1 design, a simple RR-DNS cannot make the server cluster appear as one node since clients must communicate with individual nodes for retransmissions. Moreover, if an RTSP server fails, sessions are not lost. Instead, the system reassigns the sessions to another RTSP server, with no disruption in data delivery.

We adapted the MPEG-4 file format as specified in MPEG-4 Version 2 for the storage of media blocks. This flexible-container format is based on Apple's QuickTime file format. In Yima-2, we expanded on the MPEG-4 format by allowing encapsulation of other compressed media data such as MPEG-2. This offers the flexibility of delivering any data type while still being compatible with the MPEG-4 industry standard.

To avoid bursty traffic caused by Yima-1's pause/resume transmission scheme and still accommodate VBR media, the client sends feedback to make minor adjustments to the data transmission rate in Yima-2. By sending occasional slowdown or speedup commands to the Yima-2 server, the client can receive a smooth data flow by monitoring the amount of data in its buffer.

## CLIENT SYSTEMS

We built the Yima Presentation Player as a client application to demonstrate and experiment with our Yima server. The player can display a variety of media types on both Linux and Windows platforms. Clients receive streams via standard RTSP and RTP communications.

### Client buffer management

A circular buffer in the Yima Presentation Player reassembles VBR media streams from RTP packets that are received from the server nodes. Researchers have proposed numerous techniques to smooth the variable consumption rate  $R_C$  by approximating it with a number of constant-rate segments. Implementing such algorithms at the server side, however, requires complete knowledge of  $R_C$  as a function of time.

We based our buffer management techniques on a flow-control mechanism so they would work in a dynamic environment. A circular buffer of size  $B$  accumulates the media data and keeps track of several watermarks including buffer overflow  $WM_O$  and buffer underflow  $WM_U$ . The decoding thread consumes data from the same buffer. Two schemes, pause/resume and  $\Delta p$ , control the data flow.

**Pause-resume.** If the data in the buffer reaches  $WM_O$ , the client software pauses the data flow from the server. The playback will continue to consume media data from the buffer.

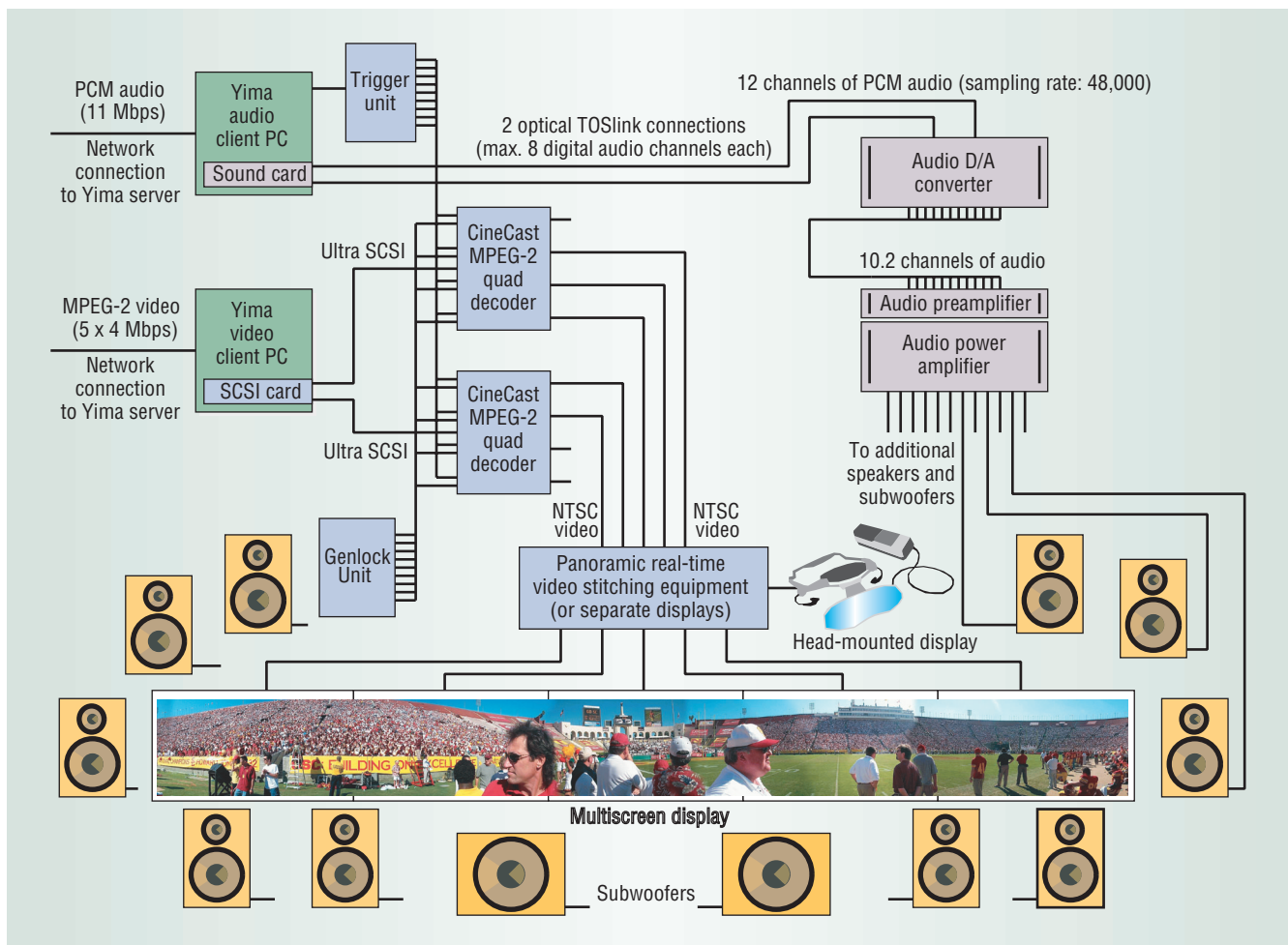
When the data in the buffer reaches the underflow watermark  $WM_U$ , the stream from the server resumes. However, the buffer must set  $WM_O$  and  $WM_U$  with safety margins that account for network delays. Consequently, if the data delivery rate ( $R_N$ ) is set correctly, the buffer will not underflow while the stream is resumed.

Although the pause/resume technique is a simple and effective design, if pause and resume actions coincide across multiple sessions, bursty traffic will become a noticeable effect.

**Client-controlled  $\Delta p$ .**  $\Delta p$  is the interpacket delivery time the schedulers use to transmit packets to the client. Schedulers use the network time protocol (NTP) to synchronize time across nodes. Using a common time reference and each packet's time stamp, server nodes send packets in sequence at timed intervals.

The client fine-tunes the delivery rate by updating the server with new  $\Delta p$  values based on the amount of data in its buffer. Fine-tuning is achieved by using multiple watermarks in addition to  $WM_O$  and  $WM_U$ , as Figure 1 shows.





**Figure 3. Panoramic video and 10.2-channel audio playback system block diagram. One Yima client renders five channels of synchronized video in a mosaic of  $3,600 \times 480$  pixels while another Yima client renders 10.2 channels of synchronized audio (0.2 refers to two low-frequency channels, or subwoofers).**

When the level of data in the client buffer reaches a watermark, the client sends a corresponding  $\Delta p$  speedup or slowdown command to maintain the amount of data within the buffer. The buffer smoothes out any fluctuations in network traffic or server load imbalance that might delay packets. Thus, the client can control the delivery rate of received data to achieve smoother delivery, prevent bursty traffic, and keep a constant level of buffer data.

### Player media types

We have experimented with a variety of media types for our Yima player. Figure 1 shows the player's three-threaded structure. The playback thread interfaces with the actual media decoder. The decoder can be either software- or hardware-based. Table 1 lists some decoders that we incorporated.

The CineCast hardware MPEG decoders from Vela Research support both MPEG-1 and MPEG-2 video and two-channel audio. For content that includes 5.1 channels of Dolby Digital audio, as used in DVD movies, we use the Dxr2 PCI card from Creative Technology to decompress both MPEG-1 and MPEG-2 video in hardware. The card also decodes MPEG audio and provides a 5.1-


channel Sony-Philips Digital Interface (SP-DIF) digital audio output terminal.

With the emergence of MPEG-4, we began experimenting with a DivX software decoder.<sup>9</sup> MPEG-4 provides a higher compression ratio than MPEG-2. A typical 6-Mbps MPEG-2 media file may only require an 800-Kbps delivery rate when encoded with MPEG-4. We delivered an MPEG-4 video stream at near NTSC quality to a residential client site via an ADSL connection.<sup>10</sup>

### HDTV client

The streaming of high-definition content presented several challenges. First, high-definition media require a high-transmission bandwidth. For example, a video resolution of  $1,920 \times 1,080$  pixels encoded via MPEG-2 results in a data rate of 19.4 Mbps. This was less of a problem on the server side because we designed Yima to handle high data rates.

The more intriguing problems arose on the client side. We integrated an mpeg2dec open source software decoder because it was cost-effective. Although it decoded our content, achieving real-time frame rates with high-definition video was nontrivial because of the high resolution. On a dual-processor 933-MHz Pentium III, we achieved



approximately 20 frames per second using unoptimized code with Red Hat Linux 6.2 and Xfree86 4.0.1 on an nVidia Quadro 2 graphics accelerator. In our most recent implementation, we used a Vela Research CineCast HD hardware decoder, which achieved real-time frame rates at data rates up to 45 Mbps.

### Multistream synchronization

The flow-control techniques implemented in the Yima client-server communications protocol synchronize multiple, independently stored media streams.

Figure 3 shows the client configuration for the playback of panoramic, five-channel video and 10.2-channel audio. The five video channels originate from a 360-degree video camera system such as the FullView model from Panoram Technologies. We encode each video channel into a standard MPEG-2 program stream. The client receives the 10.2 channels of high-quality, uncompressed audio separately.

During playback, all streams must render in tight synchronization so the five video frames corresponding to one time instance combine accurately into a panoramic mosaic of 3,600 x 480 pixels every 1/30th of a second. The player can show the resulting panoramic video on either a wide-screen or head-mounted display. The experience is enhanced with 10.2-channel surround audio, presented phase-accurately and in synchronization with the video.

Yima achieves precise playback with three levels of synchronization: block-level via retrieval scheduling, coarse-grained via the flow-control protocol, and fine-grained through hardware support. The flow-control protocol maintains approximately the same amount of data in all client buffers. With this prerequisite in place, we can use multiple CineCast decoders and a genlock timing-signal-generator device to lock-step the hardware MPEG decoders to produce frame-accurate output. All streams must start precisely at the same time.

The CineCast decoders provide an external trigger that accurately initiates playback through software. Using two PCs, one equipped with two four-channel CineCast decoders and one with a multichannel sound card, a Yima client can render up to eight synchronous streams of MPEG-2 video and 24 audio channels.

### RTP/UDP AND SELECTIVE RETRANSMISSION

Yima supports the industry-standard RTP for the delivery of time-sensitive data. Because RTP trans-

missions are based on the best-effort user datagram protocol, a data packet could arrive out of order at the client or be altogether dropped along the network. To reduce the number of lost RTP data packets, we implemented a selective retransmission protocol.<sup>11</sup> We configured the protocol to attempt at most one retransmission of each lost RTP packet, but only if the retransmitted packet would arrive in time for consumption.

When multiple servers deliver packets that are part of a single stream, as with Yima-2, and a packet does not arrive, how does the client know which server node attempted to send it? In other words, it is not obvious where the client should send its retransmission request.

There are two solutions to this problem. The client can broadcast the retransmission request to all server nodes, or it can compute the server node to which it issues the retransmission request. With the *broadcast approach*, all server nodes receive a packet retransmission request, check whether they hold the packet, and either ignore the request or perform a retransmission. Consequently, broadcasting wastes network bandwidth and increases server load.

Yima-2 incorporates the *unicast approach*. Instead of broadcasting a retransmission request to all the server nodes, the client unicasts the request to the specific server node possessing the requested packet. The client determines the server node from which a lost RTP packet was intended to be delivered by detecting gaps in node-specific packet sequence numbers. Although this approach requires packets to contain a node-specific sequence number along with a global sequence number, the clients require very little computation to identify and locate missing packets.

### TEST RESULTS

In extensive sets of experiments, Yima-2 exhibits an almost perfectly linear increase in the number of streams as the number of nodes increases. Yima-2's performance may become sublinear with larger configurations, low-bit-rate streams, or both, but it scales much better than Yima-1, which levels off early. We attribute Yima-1's nonlinearity to the increase of internodal data traffic.

We sent MPEG-4 data from the Yima servers in our lab to the public Internet via the University of Southern California campus network. The geographical distance between the two end points measured approximately 40 kilometers. We set up the client in a residential apartment and linked it to the Internet via an ADSL connection. The ADSL provider did not guarantee any minimum band-

width but stated that it would not exceed 1.5 Mbps. The raw bandwidth achieved end-to-end between the Yima client and servers was approximately 1 Mbps.

The visual and aural quality of an MPEG-4 encoded movie at less than 1 Mbps is surprisingly good. Our test movie, encoded at almost full NTSC resolution, displayed little degradation—a performance attributable to the low packet loss rate of 0.365 percent without retransmissions and 0.098 percent with retransmissions. The results demonstrated the superiority of Yima-2 in scale-up and rate control. They also demonstrated the incorporated retransmission protocol's effectiveness.

We colocated a Yima server at Metromedia Fiber Network in El Segundo, California, to demonstrate successful streaming of five synchronized video channels. Also, as part of a remote media immersion experiment (<http://infolab.usc.edu/News/NYT-RML.html>). We successfully streamed HD video at 45 Mbps from Arlington, Virginia, synchronized with 10.2-channel audio at 11 Mbps from Marina del Rey, California, to our lab at the University of Southern California.

**W**e are exploring resource management strategies across both distributed and peer-to-peer architectures in which multiple Yima clusters would exist across geographically dispersed areas.<sup>12</sup> This distribution would allow a wider range of serviceable clients. We also plan to extend the support of data types to include haptic and avatar data as part of the overall research in immersive media at USC's Integrated Media Systems Center. ■

### Acknowledgments

This research has been funded by the US National Science Foundation grants EEC-9529152 (IMSC ERC) and IIS-0082826. We thank our IMSC collaborators Chrysostomos L. Nikias, Ulrich Neumann, Alexander Sawchuk, Chris Kyriakakis, Christos Papadopoulos, and Albert Rizzo. We also thank the following students for helping with the implementation of certain Yima components: Mehrdad Jahangiri, Nitin Nahata, Sahitya Gupta, Farnoush Banaei-Kashani, and Hong Zhu.

### References

1. D.J. Gemmell et al., "Multimedia Storage Servers: A Tutorial," *Computer*, May 1995, pp. 40-49.
2. A. Bonhomme, "Survey of Video Servers," hyperlinked resource page, including bibliography, <http://www.ens-lyon.fr/~abonhomm/video/survey.html> (current May 2002; last update June 2001).
3. S. Berson et al., "Staggered Striping in Multimedia Information Systems," *Proc. 1994 ACM Sigmod Int'l Conf. Management of Data*, ACM Press, New York, 1994, pp. 79-90.
4. J.R. Santos and R.R. Muntz, "Performance Analysis of the RIO Multimedia Storage System with Heterogeneous Disk Configurations," *Proc. 6th ACM Int'l Multimedia Conference* (ACM MM 98), ACM Press, New York, 1998, pp. 303-308.
5. S. Ghandeharizadeh et al., "On Minimizing Startup Latency in Scalable Continuous Media Servers," *Proc. Multimedia Computing and Networking* (MMCN 97), SPIE-Int'l Society Optical Engineering, Bellingham, Wash., 1997, pp. 144-155.
6. J.R. Santos, R. Muntz, and B. Ribeiro-Neto, "Comparing Random Data Allocation and Data Striping in Multimedia Servers," *Int'l Conf. Measurement and Modeling of Computer Systems* (Sigmetrics 2000), ACM Press, New York, 2000, pp. 44-55.
7. R. Zimmermann and S. Ghandeharizadeh, "Continuous Display Using Heterogeneous Disk-Subsystems," *Proc. 5th ACM Int'l Multimedia Conf.* (ACM MM 97), ACM Press, New York, 1997, pp. 227-236.
8. A. Goel et al., "Scaddar: An Efficient Randomized Technique to Reorganize Continuous Media Blocks," *Proc. 18th Int'l Conf. Data Eng.* (ICDE 02), IEEE CS Press, Los Alamitos, Calif., 2002, pp. 473-482.
9. J. Hibbard, "What the \$%#@ is DivX;-)" *Red Herring Magazine*, Jan. 2001, pp. 60-64.
10. R. Zimmermann et al., "Yima: Design and Evaluation of a Streaming Media System for Residential Broadband Services," *Proc. VLDB 2001 Workshop Databases in Telecommunications* (DBTel 01), Springer-Verlag, Berlin, 2001, pp. 116-125.
11. C. Papadopoulos and G.M. Parulkar, "Retransmission-Based Error Control for Continuous Media Applications," *Proc. 6th Int'l Workshop Network and Operating Systems Support for Digital Audio and Video* (NOSSDAV 96), Springer-Verlag, Heidelberg, 1996, pp. 5-12.
12. C. Shahabi and F. Banaei-Kashani, "Decentralized Resource Management for a Distributed Continuous Media Server," to be published in *IEEE Trans. Parallel and Distributed Systems*, vol. 13, no. 6, June 2002.

*Cyrus Shahabi is an assistant professor and director of the Information Laboratory (<http://infolab.usc.edu>) in the Computer Science Department at*

the University of Southern California. He is also director of the Information Management Research Area at the Integrated Media Systems Center, an NSF Engineering Research Center at USC. His research interests include multidimensional databases, multimedia servers, and data mining. Shahabi received a PhD in computer science from USC. He is a member of the IEEE and the ACM. Contact him at shahabi@usc.edu.

**Roger Zimmermann** is a research assistant professor in the Computer Science Department at the University of Southern California and director of the Media Immersion Environment Research Area at the Integrated Media Systems Center. His interests include novel database architectures for immersive environments, video-streaming technology, cluster and distributed computing, and fault-resilient storage architectures. Zimmermann re-

ceived a PhD in computer science from USC. He is a member of the IEEE and the ACM. Contact him at rzimmerm@usc.edu.

**Kun Fu** is a doctoral candidate in computer science at the University of Southern California. His research interests include multimedia servers, real-time data distribution, and parallel computing. Fu received an MS in engineering science from the University of Toledo. Contact him at kunfu@cs.usc.edu.

**Shu-Yuen Didi Yao** is a doctoral candidate in computer science at the University of Southern California. His research interests include scalable storage architectures, multimedia servers, video streaming, and fault-tolerant systems. Yao received an MS in computer science from USC. He is a member of the ACM. Contact him at didiyao@cs.usc.edu.



# Experiments with Delivery of HDTV over IP Networks

Colin Perkins   Ladan Gharai   Tom Lehman   Allison Mankin  
USC Information Sciences Institute

15 March 2002

## Abstract

We present the design, and preliminary performance results, for a system that transports uncompressed HDTV content over IP networks. Our system is motivated by the growth in use of digital video, and the ever increasing capacity of local- and wide-area Internet links. We aim to demonstrate the feasibility of IP as a transport for very high quality video, and to highlight areas where performance bottlenecks exist and further development is needed. To this end, our system is constructed from commodity components, and was tested over existing commercial IP backbone networks. Performance was shown to be good, with the end system being the main limiting factor.

## 1 Introduction

The conversion of broadcast television from the legacy analog PAL and NTSC standards to digital format has many exciting implications. These include the possible convergence of television distribution and computer network infrastructures, allowing interactive applications, and the increase in quality possible with high definition digital formats.

To date, the different aspects of this convergence have been studied in isolation: there has been much work on the transport of compressed standard definition TV over IP, and much work defining protocols and standards for high definition TV (HDTV), but few have studied the transport of HDTV over IP. In this paper we present our initial experiments with a system to deliver production quality uncompressed HDTV over IP networks.

Why do we chose to deliver uncompressed HDTV? Several reasons, primarily to maintain image quality and reduce latency. This is most useful in a production facility, where image degradation due to repeated compression cycles is undesirable, but may also be appropriate for very high quality telepresence applications. Delivery of compressed HDTV, using existing MPEG-2 over IP standards, may be more appropriate for other applications.

The outline of this paper is as follows: section 2 covers background in HDTV technology, protocols for transport of video over IP networks and network performance. This is followed, in section 3 with a discussion of the options for protocol development, with our design being outlined in section 4. Section 5 provides preliminary performance analysis of our system, demonstrating transmission of HDTV over a wide-area IP network, with section 6 outlining directions for further development. Finally, we summarize related work in section 7, and provide conclusions.

Format	Picture Format	Ratio	Frame Rate
HDTV	1920x1080	16:9	30I, 30P, 24P
HDTV	1280x720	16:9	60P, 30P, 24P
SDTV	704x480	16:9	30I, 60P, 30P, 24P
SDTV	640x480	4:3	30I, 60P, 30P, 24P

Table 1: Picture formats for digital televisions, defined by ATSC standard A/53.

## 2 Background

The television industry is in the process of a major transformation, from standard analog PAL and NTSC systems to high definition digital formats. These new formats provide significantly higher spatial and temporal resolution, and greater colour depth, than the existing formats. The digital nature of the new formats also allows for greater integration with computer systems and networks, providing a more interactive system.

High definition TV defines a range of picture formats distinguished by frame size and rate, aspect ratio, and scanning technique (see table 1). They encompass HDTV formats with 16:9 aspect ratios and both progressive and interlaced scanning, and digital equivalents of the standard PAL/NTSC picture formats with both 16:9 and 4:3 aspect ratios. HDTV content is broadcast at 19.34Mbps, using MPEG-2 for both compression and transport [11, 10].

Local area transport of uncompressed HDTV is via coaxial cable or optical fibre, using the SMPTE-292M standard [13]. This is the universal medium of interchange between various types of HDTV equipment (e.g. cameras, encoders, VTRs, editing systems, etc.) at data rates of 1.485 Gbps. It is widely used in studios and production houses, allowing HDTV content to be delivered uncompressed through various cycles of production, avoiding the artifacts that are an inevitable result of multiple compression cycles. If wide area transport is desired, the 292M bit-stream is typically run over dedicated fibre connections, but a more economical alternative is desirable. We consider the use of IP networks for this purpose.

Standards for real-time transport of video over IP networks have reached relative maturity recently, with the dominant protocol being the Real-time Transport Protocol, RTP [20, 21]. RTP provides media framing, identifies the payload type and source, and allows for timing recovery and loss detection. It typically runs on UDP/IP networks, inheriting their limitations: unreliable, best effort delivery. Receivers use information in the RTP headers to correct for packet loss, and to reconstruct media timing. A key feature is application level framing, where the codec output is intelligently fragmented and packetized, according to a payload format, so that each RTP packet can be decoded independently [4]. This makes careful design of receivers important, since they have the primary responsibility for correct playout of media disrupted by the vagrancies of an IP network.

IP networks provide a best-effort packet delivery service. There is no guarantee that the network will not discard, duplicate, delay or mis-order packets. Applications and transport protocols using IP must adapt to these issues, abstracting the network behaviour to give a usable service. RTP applications have developed sophisticated strategies for dealing with timing jitter and packet loss [16]. It is expected that a system for delivery of HDTV over IP will use these to provide a robust service. A critical area where RTP based systems are lacking is congestion control; adapting their behaviour to fit the available network capacity. The implication here is that it is necessary to either develop congestion control for RTP or to run applications only on a network provisioned with sufficient capacity to support their needs. Of course, if it is desired to transmit uncompressed HDTV over IP, the network will need a certain capacity anyway. For this reason, we defer discussion of congestion control to section 6 and concentrate instead on the issue of finding a network that can support gigabit rate IP flows.

There are several networks that have demonstrated sufficient capacity for these experiments. Internet2's Abilene network [17] and the DARPA SuperNet testbed [24] are examples to which we have access. SuperNet has been used to demonstrate performance of 740 Mbps of single stream TCP and 957 Mbps of multi-stream TCP traffic over a cross country path [18].

Our initial testing was conducted over SuperNet between ISI East (Arlington, VA) and CMU (Pittsburgh, PA). The path includes nine hops in each direction and has an RTT of approximately 10 ms (see figure 1). We also conducted tests on a SuperNet path where the packets flowed from ISI East to ISI West (Los Angeles, CA) and back to ISI East. In this configuration, both the sender and receiver were at ISI East. This path has twenty-two hops and an RTT of approximately 67 ms.

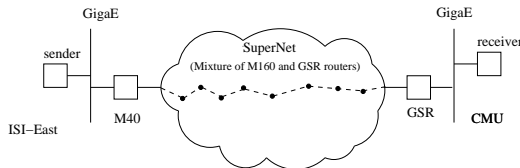


Figure 1: Data path over SuperNet from ISI-East to CMU

Our test configuration consisted of senders and receivers with gigabit Ethernet NICs connected to a switched gigabit Ethernet LAN infrastructure. The local area gigabit Ethernet connected to a site border router. Depending on the site, the border router was either a Juniper or Cisco router which connected to the wide area network via an OC48 POS connection. SuperNet uses a commercially available IP backbone for transport across the wide area. This wide area network consisted of a mix of Juniper and Cisco routers with OC48 and OC192 POS interfaces.

Prior to conducting the HDTV experiments, we first desired to ascertain that sufficient capacity was available across the wide area network. We also desired to accomplish this in a manner which would not significantly disrupt other traffic. TCP's congestion control mechanism provides a good gauge of available capacity. We used the iperf application [9] to measure both TCP and UDP bandwidth performance. Running iperf between our send machine at ISI East and the receiver at CMU, we were able to record a 702 Mbps TCP stream. Likewise we carried out the same experiment for UDP flows and were able to transfer flows in excess of 600Mbps. Performance was dependent on network load, with throughput being less at busy times.

These tests show it is possible to engineer an IP network to have low packet loss and jitter, and support gigabit rate flows. From this, we conclude that the network capacity should be available to conduct tests with HDTV over IP.

A system to transport HDTV over IP networks will use RTP as its transport, with the implication being that an RTP payload format needs to be developed for HDTV content. The options for the development of such a format are outlined in section 3, with the details of our design being presented in section 4.

### 3 Options for Transport of HDTV

A system for transport of HDTV over IP will accept a SMPTE-292M digital video signal and encapsulate it within RTP for transmission over IP. At the receiver, the SMPTE-292M signal can be regenerated, or the video can be displayed directly. There are a number of options in how this can be done, depending on the aim of the transport. If the intent is to link existing equipment the correct approach may be *circuit emulation*,



where the SMPTE-292M signal is mapped onto IP irrespective of its contents. The alternative is a *native packetization*, where an RTP payload format is defined to transport the video directly, with SMPTE-292M used only locally.

Circuit emulation provides transparent delivery of the HDTV bit-stream, suitable for input into other devices. It supports any format that SMPTE-292M supports, without having to be adapted to the details of that format. The main disadvantage is that the packetization is media unaware, and cannot optimise based on the video format. This makes circuit emulation somewhat loss intolerant.

Native packetization looks at the contents of the SMPTE-292M stream, acting on the video data within it. Hence, a native formats need to be defined for every possible video resolution, although those formats can be made more optimal. It also exposes the content to manipulation by end systems, rather than hiding it within another layer of framing.

We chose to use a native packetization, since one of our aims is to display and manipulate HDTV content on commodity workstations; we do not necessarily need to regenerate the SMPTE-292M output.

## 4 Design and Implementation

In the design and implementation of our HDTV system, our priority was to use commercial, off-the-shelf, components rather than to develop custom hardware. Accordingly, the core of our system is a high-performance PC, with gigabit Ethernet and an HDTV capture card.

The PC is a Dell Precision Workstation 620 MT with dual 1GHz Pentium III processors, running Linux 2.4.2. It has two 64 bit, 66MHz PCI slots and four 32 bit, 33MHz PCI slots. The 64 bit PCI cards are located on a separate PCI bus to the slower cards. The network interface is a 3Com 3c985 gigabit Ethernet.

For HDTV capture and playout, we use an HDstationOEM [22] card, providing SMPTE-292M input and output. This card can operate in several modes: captioning, capture and playback. We used it to capture HDTV into main memory, and to regenerate SMPTE-292M output at the receiver. Our system can also display HDTV on the workstation monitor, using a software-based decoder. The capture card supports a range of video formats, but our system uses only SMPTE-296M (1280x720 pixels, progressive scan, 60 frames per second) at this time.

The HDstationOEM card provides access to SMPTE-292M content using a FIFO API that transfers frames in order between the host and a capture/playout queue in the card's on-board memory. The API has two modes of operation: Mapped mode maps the memory on the card into the system address space. It is primarily for captioning applications, allowing small changes to be made as frames are filtered through the card. In DMA mode, the application supplies a pointer to a buffer in system memory, and the card fills that buffer with a complete frame. DMA mode is intended primarily for capture and playback applications. As noted later, we experimented with both modes of operation.

We used an updated version of the RTP library from the UCL Robust-Audio Tool [7] to provide the core network functions of our system. This is a complete RTP implementation, supporting IPv4, IPv6 and multicast. Transmission and reception were implemented as two separate programs, because the requirements of the system are such that it is not possible to transmit and receive on the same machine.



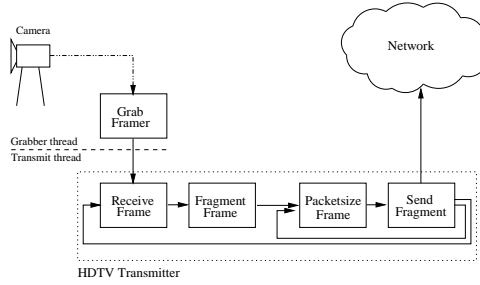


Figure 2: Block diagram of transmitter

## 4.1 Transmission

A block diagram of the transmitter is shown in figure 2. There are several parts to it: frame capture, fragmentation to match the network MTU, packetization and transmission. The capture process runs in a separate thread to the other operations, because the FIFO API provided by the grabber only supports blocking reads, that have to be overlapped with the other operations.

Once frames have been captured, they are fragmented to fit within the network MTU and transmitted in separate RTP packets. Frames are split into equal sized fragments, with an RTP payload header indicating the offset within the frame.

Our initial design smoothed transmission, spacing packets across the framing interval, rather than sending them in a burst. This proved hard to implement: the inter-packet spacing is on the order of microseconds, and system calls such as `nanosleep()` operate with a 10ms scheduling granularity under Linux, unless real-time scheduling is used. It also takes approximately  $30\mu s$  to send a packet on our system, which is comparable to the desired packet spacing of  $50\mu s$ . For these reasons, we reverted to a simple approach, sending packets back to back.

We initially used memory mapped capture, since we do not manipulate the video before transmission. Our hope was that it was not necessary to transfer the data into system memory. Rather we could calculate the fragment size, generate the RTP headers separately, and pass a pointer to the on-board memory on the capture card directly to the kernel via the `sendmsg()` system call. This performed very badly: the memory access patterns used to generate UDP packets are not optimal for the capture card.

Instead, we used DMA mode, where the capture card writes complete frames into memory. The rest of our system was unchanged: we calculate fragment sizes, and use `sendmsg()` to send the packets with a scatter/gather array, to avoid another copy in system memory. The result is that video data passes over the PCI bus twice: once from the capture card into system memory, and again from the system memory to the gigabit Ethernet card. We believe transmission performance could be greatly improved if the kernel was smart enough to use DMA for large copies in the `sendmsg()` system call.

## 4.2 Reception

A block diagram of the receiver is shown in figure 3. It operates in a classical `select()` loop, with a timeout on the order of the inter-frame time. Each iteration pulls a packet from the RTP stack, performs colour conversion if needed, and inserts the contents into a frame store at the appropriate point. If the packet

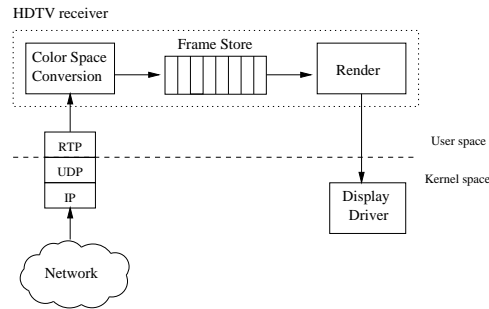


Figure 3: Block diagram of the receiver

is the last in the frame, rendering is triggered. The system also collects performance statistics and performs RTCP processing.

The first stage of reception occurs in the RTP stack. Packets are received from the kernel, validated as RTP, and then passed to the application. The RTP library used was originally written for a voice-over-IP system, and designed for flexibility rather than speed of operation. Despite this, it worked reasonably well at the rates needed for HDTV transport. The areas where performance was limited by the library included:

- Buffer allocation, since the `malloc()` system call is slow. Rather than allocate a new buffer for each packet, the library was modified to reuse buffers where possible.
- Packet validation, since the validity tests defined by RTP require a pass over the header including a number of consistency checks. This was found to take approximately 10% of the total runtime when using hardware rendering, so the library was modified to check only the RTP version number.
- Sequence number validation, as a further validity check, also uses a noticeable fraction of the system runtime, and we considered removing it. Instead, we limited our changes to a rewrite that improves the cache footprint of the code.

A number of system parameters also had to be adjusted before the full data rate could be sustained, as discussed in section 5.

Colour space conversion may be needed, depending on the display device. The HDstationOEM card can directly output the regenerated SMPTE-292M signal, but to render into a window it is necessary to convert from YUV colour space into the RGB space used by the display. Conversion is straightforward, but time consuming, since it requires arithmetic on every sample of the frame. We implement colour conversion in optimised C code, yet it takes over 90% of the total runtime when rendering into a window.

Once any necessary colour conversion has been performed, the fragment is copied to the correct location in the frame buffer. The offset is included as an RTP payload header within each packet, making this a straightforward matter. Since it is advantageous to reduce the number of copies, this step is integrated into the colour conversion code where possible.

The final packet of each frame is indicated by a marker in the RTP header, and this is used to trigger rendering. To regenerate SMPTE-292M output, the FIFO API of the HDstationOEM card is used in much the same way as for frame capture. To render into a window on the display, we use the shared memory extension of the X window system. Since rendering is triggered by receipt of the packet with the RTP marker set, our system is vulnerable to three failures:

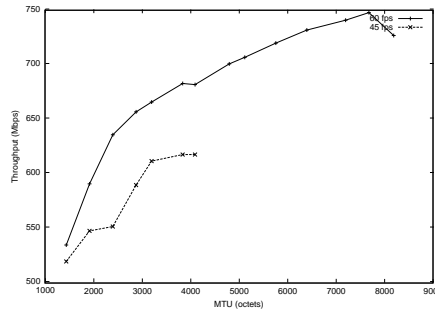


Figure 4: Effects of network MTU on throughput

- If the packet containing the marker is lost, the application will discard the frame.
- If the packet containing the marker is reordered, some fragments will be lost (they arrive after the frame has been displayed).
- If the packet containing the marker is delayed, the frame will be offset from its true playout time.

For our proof of concept system, these issues are considered an acceptable trade-off for implementation simplicity. A robust implementation would use a more sophisticated playout buffer algorithm, to smooth network jitter and to compensate for packet loss.

We have conducted a number of tests of the system performance, which we describe next, and based on these we propose a number of enhancements to our design in section 6.

## 5 Experimental Performance

### 5.1 Local area tests

Our initial trials were conducted between two hosts on the same Ethernet segment, connected via an Extreme 5i gigabit Ethernet switch. The aim was to demonstrate that our system could support HDTV delivery on an unloaded network, free from the effects of competing traffic. The tests were successful: when correctly tuned, our implementation is loss free in the local area tests. The tuning process was significant, however, requiring adjustments to the network maximum transfer unit (MTU), socket buffer size and network driver.

With the default 1500 octet MTU, the system throughput is 535 Mbps. This is insufficient for our needs, but gigabit Ethernet interfaces support the jumbo-frames extension, allowing us to increase the MTU to 9000 octets. Increasing the MTU affects throughput as shown in figure 4: larger MTU sizes result in higher throughput. The increase is due to the reduction in the header processing overhead relative to the amount of data, and the reduction in interrupt load on the host. We chose a 4470 octet MTU for our tests, even though that does not give best performance, since it is the maximum supported by the wide area network, and we desired to compare our local and wide area results.

The default 64k socket buffer was insufficient, and caused packet loss in the receiving host. Experience shows that the buffer may need to be large enough to store the packets for an entire frame of video. This because the receiver is not able to process packets continually: there are some periods when it is busy

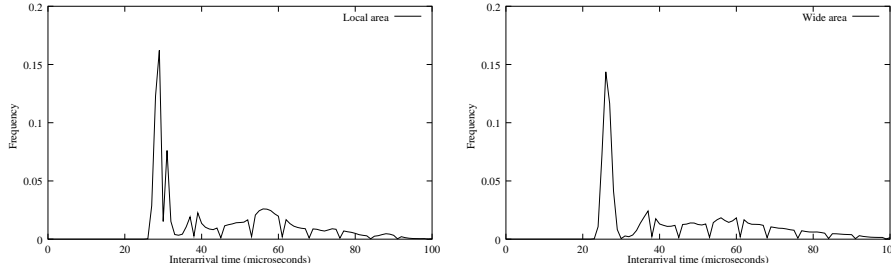


Figure 5: Inter-packet timing at the receiver

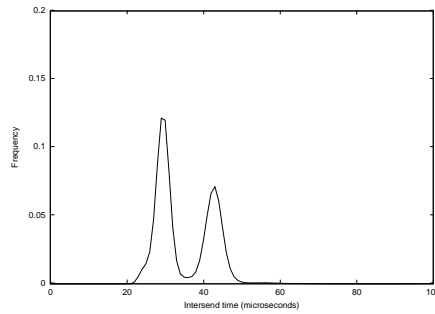


Figure 6: Inter-packet timing at the sender

processing the video, and cannot receive packets. This is a particular problem with our software decoder, since colour conversion takes a significant amount of time. Multithreading the receiver is expected to reduce the buffering requirements, since it will allow concurrent packet reception and media processing/decoding on dual processor systems.

The gigabit Ethernet driver performs interrupt coalescing and checksum offloading, and had delayed packet notification enabled. Adjusting these parameters does not appear to significantly affect performance, and after much experimentation we settled on the default values.

With these changes in place, the system can sustain a transfer rate of 615 Mbps, with no packet loss. This allows us to send 1280x720 pixel video at 45 frames per second, using 8 bits per colour component. Once packet loss was eliminated, we made two sets of measurements relating to the network timing jitter: the inter-packet timing and relative transit delay.

Figure 5 plots inter-packet timing against relative frequency of occurrence, for both local and wide-area tests. It should be compared with Figure 6, which shows timing measured at the sender. As can be seen, the inter-packet timing is strongly bi-modal, a result which surprised us since the transmitter sends the packets that comprise each frame in a tight loop (there is a much smaller peak at the location of the inter-frame interval, which is not visible in the figure). The bimodality seems due to the sender blocking in the transmit call, perhaps due to limited buffering in the network card. If the on-board buffer is full, we expect the system blocks until the packet is sent, causing some packets to be delayed. The effect of network transit is to smear this second peak out in time. The receiver sees the initial peak in the inter-packet timing, with the same interval as the sender, but a broader tail to the distribution.

The relative transit delay, the difference between the arrival time in RTP clock units and the send time in the same units, is illustrated in Figure 7. We note variation of approximately 10ms, equal to the Linux

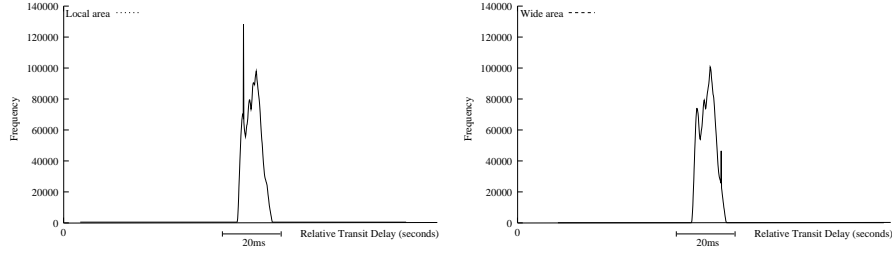


Figure 7: Relative transit delay

Loss event duration	Frequency
No loss	24697400
Single packet	85797
Two consecutive packets	587
Three consecutive packets	7
Four or more packets	0

Table 2: Observed packet loss rates

scheduling interval, and believe these measurements are heavily influenced by the time taken to wake the receiver on arrival of the first packet in a frame, and that such large variation in network transit time is largely a measurement artifact. Further study is needed to confirm this result.

Surprisingly, our tests also revealed the presence of a small amount of packet reordering between hosts on the same Ethernet segment. Typical measurements showing approximately 1 in 10000 packets being delivered out of order. Reordering persists when the two hosts are connected back to back using cross-over fibre, and can also be demonstrated with the iperf tool [9]. We have no good explanation for this, but suspect a race condition in the Linux 2.4 kernel, triggered by our use of dual processor systems.

Video quality was excellent during the tests, although 45 frames per second does not result in optimally smooth motion (every 4th frame of the original is dropped, so the frame timing is not uniform).

## 5.2 Wide area tests

We conducted a number of wide area tests of our system, using various paths across the DARPA SuperNet testbed. The first factor evaluated was packet loss on the wide-area network path. This was partly to ensure that we were not causing network congestion, and partly to determine the effects of packet loss on the video quality. Packet loss was difficult to measure, since the network commonly operated without loss. Table 2 shows typical measurements when the network was loaded: approximately 0.3% of packets were lost, with most loss events being of isolated packets. More common was the case where no loss was observed.

The distribution of inter-packet arrival times and relative transit delay for the wide area network path, shown in Figures 5 and 7, is almost identical to that for the local area tests. The network is lightly loaded, and hence there is no significant queueing jitter to impact the packet timing.

As to be expected, some small degree of packet reordering was present on the wide area path. Typical results showed 0.05% of packets delivered out of order, with the vast majority of reordering events being

of adjacent packets. In rare cases, we observed packets being delivered two or three out of sequence. This degree of reordering is not unusual, similar values have been reported by [2, 3, 15]

Video quality for the wide-area tests was subjectively identical to that observed in the local tests.

In addition to the closely monitored tests over DARPA's SuperNet, we also demonstrated the system at the SuperComputing 2001 conference in Denver, November 2001. In this demonstration, the system was run over a path from Washington D.C to Denver. The network path for this demonstration utilised Internet2's Abilene and the Washington D.C. area MAX gigapop. This path was 10 router hops with a RTT of approximately 43 ms. The backbone links along this path were OC48 POS and there was no advance resource reservation. We have no formal measurements of the system performance over this path, but informal observations of the system showed negligible packet loss and jitter. Very high quality video was received, with no apparent problems, for a period of several hours.

## 6 Limiting Factors and Future Directions

Our experiments were conducted using 615 Mbps media streams, comprising 1280x720 pixel images at 45 frames per second with 8 bits per colour component. This is insufficient for true uncompressed HDTV, which requires 850 Mbps to increase the frame rate to 60 frames per second, and 1.03 Gbps for full colour.

PCI bus contention appears as the main limiting factor. We initially put the HDTV capture card and the gigabit Ethernet into the two 64 bit/66 MHz PCI slots on the PC, but tests showed that performance *increased* when we moved the Ethernet card onto a slower 32 bit/33 MHz PCI slot. Investigation showed that the fast PCI slots shared a single bus, distinct from that used by the slower slots, leading us to believe that contention on the bus was an issue. We have under development a system using a PC with dual 64 bit/66MHz PCI bus architecture, which we expect to reach the 850Mbps data rate needed for full frame-rate HDTV, subsampled to 8 bits per component.

To support the full colour depth – 10 bits per component – we need a faster network interface, for example a PCI-based OC-48 interface. Our initial experiments with such interfaces have been disappointing, with the available cards being unable to exceed the transfer rates achievable using gigabit Ethernet. Use of dual gigabit Ethernet may also be possible, but we may again run into the limitations of the PCI bus.

An alternative to faster networks may be use of lossless video compression, to reduce the bandwidth requirements of the system. This is an area to be explored in future, although it is not clear that commodity systems can compress gigabit rate streams in real-time.

Memory bandwidth also limits performance; the system has been refactored several times to reduce the number of copies, increasing performance. This is especially an issue for the software decoder, rendering into a window, due to the need for colour conversion. Use of MMX extensions is expected to help, as will off-loading conversion using the hardware acceleration available in some display adaptors. Interrupt processing overheads are also a factor, evidenced by increased throughput when larger packets are used. This is an area where wide area networks limit performance, since they limit the MTU to 4470 octets.

Our implementation has a simple playout routine, and does not deal well with jitter or loss around frame boundaries. We plan to implement an adaptive playout buffer, to compensate for jitter and to correct frame playout time. Once this is done, we plan to study more sophisticated error correction and concealment algorithms. At present, packet loss is concealed by repeating part of the previous frame to cover the missing data. If frames are buffered before playout, it will be possible to use some form of FEC (e.g. [19]) or retransmission to correct lost packets.

Congestion control is a serious issue for high rate UDP applications on the current Internet. Our implementation is not currently congestion controlled, raising the issues of fairness to other traffic and potential congestion collapse of the network. Before we deploy our system outside a controlled environment, we need to implement some form of rate limiting or congestion control. The TFRC protocol [5] might be an appropriate means of congestion control, but more work is needed to implement and evaluate this.

Our implementation uses a simplistic RTP payload header, consisting only of the fragment offset within a frame (the fragment length being inferred from the packet length). A more general payload format for uncompressed video, better preserving frame metadata should be defined. The RTP payload format for BT.656 video [25] may be suitable as a base design, although it will need extension for HDTV formats.

## **7 Related work**

A product from 2netFX [1] delivers compressed HDTV over IP. The system uses MPEG-2 compression at 19.2Mbps using the standard RTP payload [8]. The use of compression adds latency and makes this system unsuitable for environments where video editing is performed, or where full quality is needed.

The University of Washington have demonstrated a system for transport of HDTV over IP [14]. This system uses Sony HDCAM compression at 270 Mbps. This is a proprietary production quality compression scheme, supporting a limited number of edit cycles without significant quality degradation.

A prototype developed by Tektronix [23] uses custom hardware to deliver HDTV over an OC-48 POS interface. The system performs circuit emulation of SMPTE-292M over IP at 1.5 Gbps using an RTP payload format [6] developed in conjunction with the University of Washington and ourselves. This system was also demonstrated at the Super Computing 2001 conference.

Most similar to our work is the system built by NTT Laboratories, which was demonstrated in Tokyo, October 2001 [12]. This system is built around a multi-processor PC running Linux, with a commercial HDTV capture card, but uses a custom network interface.

These latter two systems suffer from being implemented using custom hardware. This makes them expensive and inflexible, compared to a system built using off the shelf components. Their advantage is that they have better performance at present, although we expect that Moore's law will close this gap rapidly.

## **8 Conclusions**

We have successfully demonstrated a prototype system for transport of uncompressed HDTV over IP networks, which we believe is the first built using commodity components. The system currently supports SMPTE-296M format pictures at a reduced rate of 45 frames per second, with colour sub-sampled to 24 bits. An enhanced version is under development, which we expect to support the full 60 frames per second, and we further plan to extend the system to deliver full quality uncompressed video.

There are a number of challenges to supporting full uncompressed HDTV, primarily due to limitations of the end system. We have described a number of areas where performance may be improved; further work will implement some of these ideas. Many of these techniques are also valid for high bit rate compressed video, transport of HDTV over IP provides an appropriate tested, but is not the only application that may benefit from this work.

## Acknowledgements

This work is supported by DARPA ITO under the Next Generation Internet program, and by hardware donated by Intel corporation. The MAX gigapop and Abilene NOC provided assistance in conducting the wide-area tests. Juniper Networks loaned equipment for our demonstration at SuperComputing 2001.

## References

- [1] 2netFX. Thundercastip advanced media server. <http://www.2netfx.com/>.
- [2] J. C. R. Bennett, C. Partridge, and N. Shectman. Packet reordering is not pathological network behavior. *IEEE/ACM Transactions on Networking*, 7(6):789–798, December 1999.
- [3] E. Blanton and M. Allman. On making TCP more robust to packet reordering. *ACM Computer Communication Review*, January 2002.
- [4] D. D. Clark and D. Tennenhouse. Architectural considerations for a new generation of protocols. *Computer Communications Review*, 20(4):200–208, September 1990.
- [5] S. Floyd, M. Handley, J. Padhye, and J. Widmer. Equation-based congestion control for unicast applications. In *Proceedings of ACM SIGCOMM 2000*, Stockholm, Sweden, 2000.
- [6] L. Gharai, G. Goncher, C. S. Perkins, D. Richardson, and A. Mankin. RTP Payload Format for SMPTE 292M. Internet Engineering Task Force, July 2001. Work in progress.
- [7] O. Hodson and C. S. Perkins. Robust-audio tool, version 4. <http://www-mice.cs.ucl.ac.uk/multimedia/software/rat/>.
- [8] D. Hoffman, G. Fernando, V. Goyal, and R. Civanlar. RTP Payload Format for MPEG1/MPEG2 Video, January 1998. RFC 2250.
- [9] Iperf. <http://dast.nlanr.net/Projects/Iperf/>.
- [10] ISO/IEC. Generic coding of moving pictures and associated audio information: Systems, 1996. ISO/IEC 13818-1.
- [11] ISO/IEC. Generic coding of moving pictures and associated audio information: Video, 1996. ISO/IEC 13818-2.
- [12] NTT Innovation Laboratories. Uncompressed HDTV transmission system over the internet. NTT Press Release, October 2001. <http://www.ntt.co.jp/news/news01e/0110/011026.html>.
- [13] Society of Motion Picture and Television Engineers. Bit-serial digital interface for high-definition television systems, 1998. SMPTE-292M.
- [14] University of Washington. Internet hdtv. <http://www.washington.edu/hdtv/>.
- [15] V. Paxson. End-to-end internet packet dynamics. *IEEE/ACM Transactions of Networking*, 7(3), June 1999.
- [16] C. S. Perkins, O. Hodson, and V. Hardman. A survey of packet loss recovery techniques for streaming media. *IEEE Network Magazine*, September/October 1998.
- [17] The Internet2 project. <http://www.internet2.edu/>.
- [18] SuperNet Land Speed Record. <http://www.ngi-supernet.org/wan-speed.html>.
- [19] J. Rosenberg and H. Schulzrinne. An RTP payload format for generic forward error correction, December 1999. RFC 2733.
- [20] H. Schulzrinne. RTP profile for audio and video conferences with minimal control, January 1996. RFC 1890.
- [21] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RTP: A transport protocol for real-time applications, January 1996. RFC 1889.
- [22] DVS Digital Video Systems. Hdstationoem card. <http://www.dvs.de/english/products/HDStationPRO/HDStationOEM.htm>.
- [23] Tektronix. Universal network access system. <http://www.tektronix.com/Measurement/commtest/darpa/darpa.html>.
- [24] The DARPA SuperNet testbed. <http://www.ngi-supernet.org/>.
- [25] D. Tynan. RTP Payload Format for BT.656 Video Encoding, October 1998. RFC 2431.



# Applied Techniques for High Bandwidth Data Transfers across Wide Area Networks

Jason Lee, Dan Gunter, Brian Tierney  
Computing Sciences Directorate  
Lawrence Berkeley National Laboratory  
University of California, Berkeley, CA 94720  
{jrlee,dkgunter,bltierney}@lbl.gov

Bill Allcock, Joe Bester, John Bresnahan, Steve Tuecke  
Mathematics and Computer Science Division  
Argonne National Laboratory  
9700 South Cass Ave., IL, 60439  
{allcock,bester,breshaha,tuecke}@mcs.anl.gov

## Abstract

*Large distributed systems such as Computational/Data Grids require large amounts of data to be co-located with the computing facilities for processing. Ensuring that the data is there in time for the computation in today's Internet is a massive problem. From our work developing a scalable distributed network cache, we have gained experience with techniques necessary to achieve high data throughput over high bandwidth Wide Area Networks (WAN). In this paper, we discuss several hardware and software design techniques and issues, and then describe their application to an implementation of an enhanced FTP protocol called GridFTP. We also describe results from two applications using these techniques, which were obtained at the Supercomputing 2000 conference.*

## 1.0 Introduction

Large distributed systems such as Computational/Data Grids [10] require large amounts of data to be co-located with the computing facilities for processing. Ensuring that the data is there in time for the computation to start in today's Internet is a massive problem. At LBNL we developed a high-performance striped data cache called the Distributed Parallel Storage System (DPSS)[29]. The idea behind the striped server is shown in Figure 1: several disks on several hosts operate in parallel to supply a very high-speed data stream to one or more clients. The DPSS was specifically optimized for access to large data objects by remote clients over a wide area network.

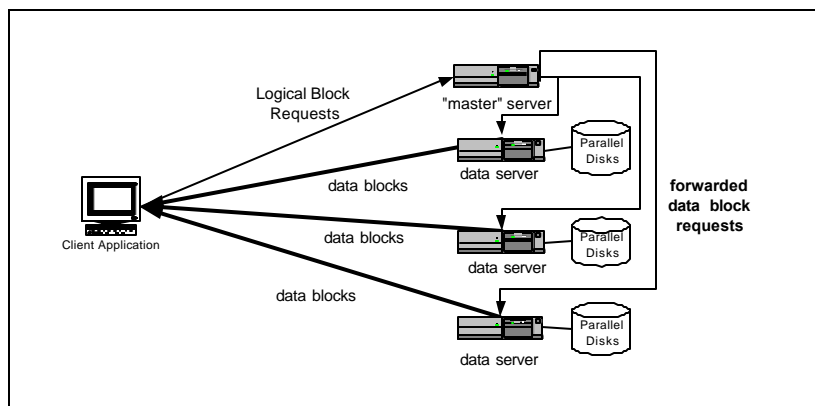


Figure 1: Striped Data Server Architecture

In the course of designing and using the DPSS we have gained experience with some techniques to achieve high data throughput over the WAN. In addition, some general lessons have been learned on the proper configuration of hardware. The following is a brief summary of these techniques, several of which are described in more detail in later sections of this paper:

- use parallel everything (i.e.: servers, disks, SCSI controllers, network cards)
- use tuned TCP buffers, optimally set for each client
- use parallel streams, but only if the receive host is powerful enough
- use asynchronous and overlapped network and disk I/O
- no user-space data copying allowed (manipulate all data buffers via pointers)
- provide a simple client API that is as similar to the POSIX I/O interface as possible.

LBNL and ANL worked together to abstract these techniques from the DPSS and apply them to the development of a high-throughput striped server which interfaces to the outside world using an enhanced version of the File Transfer Protocol (FTP)[21] called GridFTP [11]. In this paper, we will use both the DPSS and GridFTP to illustrate implementation issues related to the techniques, and the preliminary results from a very high bandwidth WAN test of both storage servers at SuperComputing 2000.

## 1.1 Motivation

In scientific computing environments, small clusters of PC's running Linux are becoming more common, and these will likely become an important part of the future data intensive computing landscape. Large PC clusters running data intensive applications will likely use a Storage Area Network (SAN) with multiple high performance Redundant Array of Inexpensive Disk (RAID)[20] systems as data servers. Although considerably cheaper than the previous large RAID-based solutions, this type of hardware configuration is still relatively expensive because it requires the purchase of Fibre Channel switches and interface cards[7]. For large clusters the costs will be amortized over many nodes, and perhaps their required performance will be unattainable by other means, but sites with small clusters will probably not be able to afford this.

Although small clusters most likely cannot afford an expensive SAN/RAID system, they do need better performance than that provided by the typical solution today, which is a medium strength (e.g. 4 CPU) NFS server connected to a single RAID system. This is certain to be a bottleneck for some data intensive applications. In order to attain the necessary performance at a low cost, we argue that a set of striped servers composed of commodity hardware and software, and running over the existing high-speed LAN, should be used. The scalability and price/performance ratio of striped commodity servers make them an excellent solution for this environment

For example, a one terabyte data set might be staged from a tape system such as HPSS to a striped cache system. If the data is striped across 8 data cache hosts, then a 32 node cluster would receive up to 8 times more I/O bandwidth than it would using a single data server.

In addition to high-throughput from data cache to cluster, high-throughput from data cache to data cache across a WAN is also very important. There are several scientific research communities that need the ability to copy and/or replicate data quickly between disk caches at multiple sites [5][14]. A striped data server that is optimized for WAN data transfers is ideal for this environment.

## 2.0 Techniques for High Performance

Several techniques for achieving high-performance from storage servers in a WAN environment are presented below. It is important to note that these techniques are not additive, but rather complementary. Applying a single technique may have little or no effect, because the absence of any one of the techniques can create a bottleneck.

### 2.1 Tuned TCP Buffers

The standard transport layer in use today is the Transport Control Protocol (TCP) [27]. TCP uses what it calls the "congestion window", or CWND, to determine how many packets can be sent before waiting for an acknowledgement. The larger the congestion window size, the higher the throughput. This follows directly from Little's Law[17] that (average throughput)\*(delay) = window size. The TCP "slow start" and "congestion avoidance" algorithms determine the size of the congestion window [16]. The maximum congestion window is proportional to the amount of buffer space that the kernel allocates for each socket. For each socket, there is a default size for this buffer, which can be changed by the program (using a system library call) just before opening the socket. There is also a kernel-enforced maximum buffer size. The buffer size can be adjusted for both the send and receive ends of the socket.

To get maximal throughput it is critical to use optimal TCP send and receive socket buffer sizes for the link you are using. If the buffers are too small, the TCP congestion window will never fully open up. If the buffers are too large, the sender can overrun the receiver, and the TCP window will shut down. The optimal buffer size of a link is (bandwidth) \* (round trip time (RTT)), where RTT equals twice the one-way delay on the link. For an explanation of why this is true, see [22] and [29].

For example, if your RTT is 50 ms, and the end-to-end network consists of all 100BT ethernet or higher, the TCP buffers should be 0.05 sec \* 10 MB/sec = 500 KBytes. Two TCP settings need to be considered: The **default** TCP send

and receive buffer size, and the **maximum** TCP send and receive buffer size. Note that most of today's UNIX operating systems ship with a maximum TCP buffer size of only 256 KB (and the default maximum for Linux is only 64 KB!). The maximum buffer size need only be set once. However, since setting the default TCP buffer size greater than 128 KB will adversely affect LAN performance, the UNIX *setsockopt* call should be used in your sender and receiver to set the optimal buffer size for the link you are using. For more information on setting the default, maximum, and optimal TCP buffers, consult the LBNL "TCP Tuning Guide"[26].

## 2.2 Parallel Streams

The design of TCP has been influenced much less by the high-performance community than by the demands of the Internet, and in particular by the need to enforce fair sharing of precious network resources. For this reason, TCP's behavior is unnecessarily conservative for data-intensive applications on high bandwidth networks.

TCP probes the available bandwidth of the connection by continuously increasing the window size until a packet is lost, at which point it cuts the window in half and starts "ramping up" the connection again. The higher the bandwidth-delay product, the longer this ramp up will take, and less of the available bandwidth will be used during its duration. When the window of the bottleneck link is large enough to keep the pipe full during the ramp up, performance does not degrade. However this requires large buffers on all intervening routers. Furthermore, when there is random loss on the connection, it has been shown [16] that the link utilization is proportional to  $q(\mu\tau)^2$ , where  $q$  is the probability of loss and  $\mu\tau$  is the bandwidth-delay product.

In order to improve this situation where the network becomes the bottleneck, parallel streams can be used. This technique is implemented by dividing the data to be transferred into  $N$  portions and transferring each portion with a separate TCP connection. The effect of  $N$  parallel streams is to reduce the bandwidth-delay product experienced by a single stream by a factor of  $N$  because they all share the single-stream bandwidth ( $\mu$ ). Random packet losses for reasonable values of  $q$  ( $<0.001$ ) will usually occur in one stream at a time, therefore their effect on the aggregate throughput will be reduced by a factor of  $N$ . When competing with connections over a congested link, each of the parallel streams will be less likely to be selected for having their packets dropped, and therefore the aggregate amount of potential bandwidth which must go through premature congestion avoidance or slow start is reduced. It should be noted, however, that if the bandwidth is limited by small router buffers in the path, all the streams are likely to experience packet loss in synchrony (when the buffer fills, arriving packets from each stream are all dropped) and thus gain little advantage over a single stream.

Experience has shown that parallel streams can dramatically improve application throughput, (see [23] and [29]), and can also be a useful technique for cases where you don't have *root* access to a host in order to increase its maximum TCP buffer size. However, parallel streams can drastically reduce throughput if the sending host is much faster than the receiving host. For example, we have seen a 50% loss in aggregate throughput of 2 streams versus 1 stream on a Linux 2.2.14 receive host with a NetGear 1000BT card using a multi-threaded receiver.

## 2.3 Striped Disks and Servers

In order to aggregate the potential throughput of numerous hosts, each with one or more disk controllers and several disks per controller (see Figure 2), the data being transferred should be subdivided into "stripes" and spread evenly across the servers and disks. Different software or hardware systems may be responsible for striping at different levels of the storage hierarchy. For example, a RAID system may stripe across the file system on a host, while a separate server stripes across all the hosts. The important point is to make sure that the striping occurs at all levels. Thus the disks can saturate the disk controllers, the disk controllers can saturate the network interface card (NIC), and the NICs can saturate the router.

Placement algorithms affect the parallelism of the system. A good stripe placement for sequential operations, i.e. for data access patterns with a high temporal locality, is round-robin. For random-access data sets with a high spatial locality (i.e. several widely spaced areas of the dataset are accessed in parallel), partitioning the file into one contiguous sequence of stripes per disk may improve performance.

The size of the stripe must balance the need to evenly distribute the data across the storage resources (smaller is better) with the need to perform efficient low-level I/O to both disk and network (bigger is better). Although the exact size of the stripe may not be critical, small numbers might lead to a horrible bottleneck. For example, see the results in Table 1. For random access, large disk reads can dramatically improve disk throughput.

It is important to have enough parallel disks on each server to saturate the network under non-sequential access patterns. For example, if you are using a 64KB stripe size and the same SCSI disk that was used for the results in Table

1, and your server network card has a maximum throughput of 40 MB/s, then you will need 8-9 parallel disks to saturate the NIC.

Table 1

Access Method	1 KB blocks	8 KB blocks	64 KB blocks	128 KB blocks
sequential	18.2 MB/s	18.5 MB/s	22.7 MB/s	22.7 MB/s
random	.08 MB/s	2.2 MB/s	4.6 MB/s	8.0 MB/s
speedup	231	13.3	4.9	2.7

### 3.0 Hardware Configuration Issues

The simple employment of these techniques in an application does not guarantee the absence of performance bottlenecks. Interaction patterns across the hardware can play an important role in creating performance problems in an application. In the following sections we will try to examine and summarize some of the technical issues that were encountered during the development of the techniques.

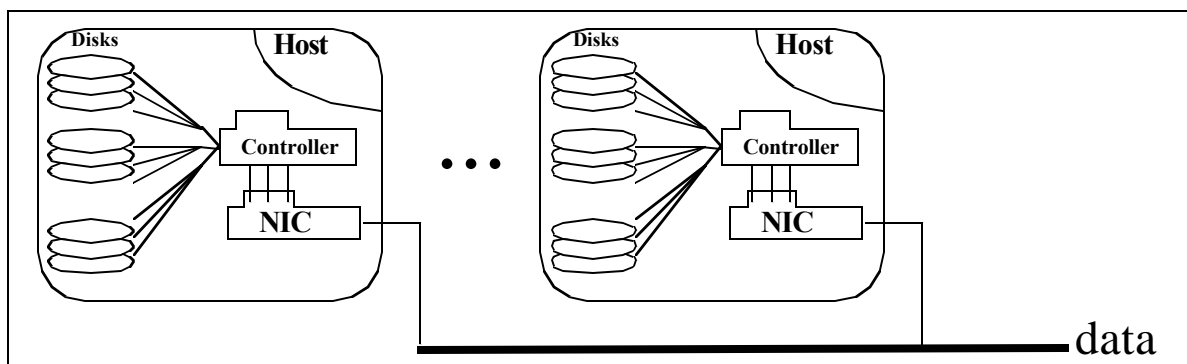


Figure 2: Disk, controller, and network parallelism

### 3.1 Disks and Controllers

In today's high-speed computing environment it is important to ensure that both your disks and controllers are not simply fast enough, but well matched (properly configured and tuned). Improperly configured hardware can cause unnecessary thrashing of system resources. In a bottom up approach to tuning the I/O subsystem one should first start by testing the disks, then the controllers and then move up through the system till you reach host adapters.

The disks should be of equal size and speed. The state of the system is limited by its slowest component, therefore a slower disk will constrain the performance of a striped file system. Smaller disks will skew the striping performance because the larger disk will be accessed more often, instead of striping the data equally across all the disks.

Once a decision has been made on the which hardware to use, each component of the system should be tested. First, test the speed of a single disk, in isolation. Next, add a second disk and check the speed of the two disks being accessed simultaneously. With the addition of each disk, the aggregate speed should increase by the speed of a single disk. Once the aggregate throughput stops increasing, the limit of the disk controller has probably been reached; if the throughput is not enough to saturate the NIC, another disk controller will need to be added, and the process of adding disks should continue.

For example, with a NIC on Gigabit Ethernet (~300Mb/s), a SCSI controller at 160Mb/s, and SCSI disks as shown in Table1 above, to saturate the NIC for random access reads there will have to be 2 controllers and at least 2 disks per controller.

### 3.2 Networking and Processors

As the network, disk and memory speeds all increase, the speed, power and number of CPU's in the system become an increasingly important factor. Many of the newer high-speed network cards now require a significant amount of CPU power to drive them. The number of interrupts that are delivered to the OS when running at gigabit speeds can overwhelm slower CPU's. There are several ways to lower the load on the CPU:

- Interrupt coalescing; the network card packages several TCP packets together before interrupting the OS

- TCP checksumming on the network card, instead of in software on the host computer
- Larger Maximum Transmission Unit (MTU)

All three of these techniques attempt to accomplish the same goal: reduce the per-packet processing overhead. From [6], “Smaller frames usually mean more CPU interrupts and more processing overhead for a given data transfer size. Often the per-packet processing overhead sets the limit of TCP performance...”.

It should be noted that not all gigabit network interfaces support all these options, or will interoperate with other cards or switches. Most notably when using a larger MTU (sometimes referred to as a Jumbo Frame) packets may not be able to cross some networks or interoperate with some switches.

We tested several different vendors’ cards, and found a large degree of variance in how they performed, depending on variables such as the PCI bus width (64 vs. 32bit), how much memory was on the card, what driver/OS were used to drive the card. For instance, by changing from Linux kernel version 2.2 to version 2.4, our local area *iperf* throughput values rose from approximately 320 Mb/s to just over 500 Mb/s using the same hardware. In conclusion, one should always test out the specific cards in the environment that will be used in production.

### 3.3 Implementation

We have made two very different implementations of the various techniques that we have described in this paper. The first one is the DPSS, which uses a custom API and was created especially for use in a WAN environment. Secondly, groups at ANL and LBNL worked together to design a system that uses techniques learned while developing the DPSS and applies these techniques to build a more general purpose high-performance FTP server. This ‘enhanced’ version of FTP, called GridFTP, supports striped servers, parallel streams, and tuned TCP window buffers, and was developed in conjunction with ANL’s data transfer libraries [8].

Both the DPSS and the GridFTP server operated on an identical hardware configuration. Typical striped server implementations consist of several low-cost Unix workstations as data block servers, each with several disk controllers, and several disks on each controller. A four-server system with a capacity of one Terabyte (costing about \$10-\$12K in late-2000) can produce throughputs of over 70 MB/s by providing parallel access to 20-30 disks.

### 3.4 DPSS

The main features of the DPSS are described in the first section and include but are not limited to: highly parallel, tunable TCP buffers, and asynchronous I/O. During DPSS usage, requests for blocks of data are sent from the client to the “DPSS master” process, which determines which “DPSS block servers” the blocks are located on, and forwards the requests to the appropriate servers. The server then sends the block directly back to the client.

The application interface to the DPSS is through either a low level “block” API, or a higher level POSIX-like API. The data layout on the disks is completely up to the application, and the usual strategy for sequential reading applications is to write the data round-robin, striping blocks of data across the servers. The DPSS client library is multi-threaded, where the number of client threads is equal to the number of DPSS servers. Therefore, the speed of the client is scaled with the speed of the server, assuming the client host is powerful enough.

### 3.5 GridFTP

GridFTP consists of extensions to the FTP protocol to provide features necessary for a Grid environment. Use of a common protocol provides interoperability: GridFTP can communicate with any existing FTP server, or any new implementation that follows the extended FTP protocol.

Most current FTP implementations support only a subset of the features defined in the FTP protocol and its accepted extensions. Some of the seldom-implemented features are useful to Grid applications, but the standards also lack several features Grid applications require. We selected a subset of the existing FTP standards and further extended them, adding the following features: Security (both Grid Security Infrastructure (GSI) and Kerberos support), Parameter set/negotiate, which allows interoperability with existing FTP implementations, Third party transfers (server to server), parallel transfers (multiple TCP streams per transfer), striped transfers (multiple host to multiple host), partial file transfers, and flexible reliability / recovery functionality via a plug-in interface. The actual protocol extensions are beyond the scope of this paper, but a proposed draft submitted to the Grid Forum may be reviewed at <http://www.gridforum.org>.

GridFTP can be used for bulk data transfer as in this application, or can be used as a data access mechanism with semantics very similar to Unix open/close/seek/read/write from an application perspective. For this implementation, the physical mechanism employed is essentially a map of the file into a pool of 64KB blocks. The distribution of these blocks is user selectable and may be either partitioned (the file is divided into  $n$  pieces and one piece is stored on each node, where  $n$  is the number of nodes) or round robin.

To initiate a transfer, third party in this case, a control connection is established between the application and the master server at each site. These master servers form control connections to the back end servers. Once these connections are established, a fairly standard FTP protocol exchange takes place between the application and the master servers. The master forwards the commands to the back end servers and condense the individual responses into a single response that is then sent to the application. The back end servers check a local database to determine which blocks, if any, they have, they then establish data connections with the appropriate source/destination server, with the specified level of parallelism, and execute the transfer.

## 4.0 Results

Gaining access to the next generation of high-speed networks in order to explore the techniques outlined above is difficult. We were able to participate in a contest called the “Bandwidth Challenge” at SuperComputing 2000, which used a time-shared OC-48 (2.4 Gb/s) network path over NTON [19] and SuperNet [24] from LBNL in Berkeley, CA to the conference show floor in Dallas, TX, as shown in [3]. Both the DPSS -- as a server for an application called Visapult [3] -- and GridFTP participated in the contest, and each had exclusive access to this network path for a one hour run. We were able to monitor the router in Berkeley during both runs. In this section, we will briefly describe each application, and analyze their results. Due the transient nature of the network, we did not have time to run more controlled experiments, so the degree to which these results characterize general performance characteristics of either the DPSS or GridFTP is uncertain, and will be the subject of future work.

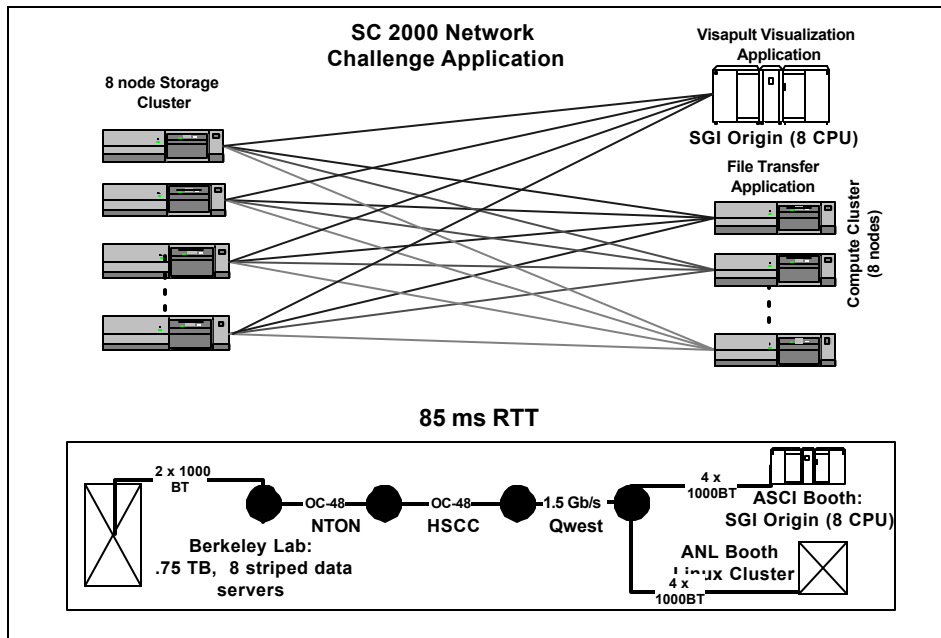


Figure 3: SC2000 Network Challenge Application Configuration

The servers for both the DPSS and GridFTP were identical hardware, which consisted of: 4 dual-processor Linux boxes and 4 single-processor Solaris boxes, all with SCSI disks and a single controller, running over Gigabit Ethernet.

### 4.1 Visapult / DPSS Results

The LBNL entry in the challenge was a visualization application, Visapult, that transforms scientific simulation data in parallel on a compute node, then transmits it in parallel over the network for rendering. The dataset, 80GB in size, was stored on the DPSS at LBNL and the compute cluster, an 8-processor SGI Origin with 4 Gigabit Ethernet interfaces, was in Dallas. This process of parallel reading of a large dataset from a distant location while transforming it on a powerful compute node is common in high energy physics (HEP) applications. During the course of the contest, background DPSS *get* operations were run to use up the spare DPSS bandwidth (roughly 500 Mb/s) due to a bottleneck at Visapult’s rendering engine. The DPSS obtained a peak of 1.48 Gb/s and sustained throughput of 582 Mb/s, as measured at the ingress router to the show floor.

A graph of 5-second polls of the router packet counts is shown in Figure 4. The graph shows the router throughput over time. Because the test had 64 streams (8 nodes x 8 processors on the SGI), the CWND was only about 200KB, instead of the 13MB a single stream would have required. This allowed us to better utilize the network and adapt to it.

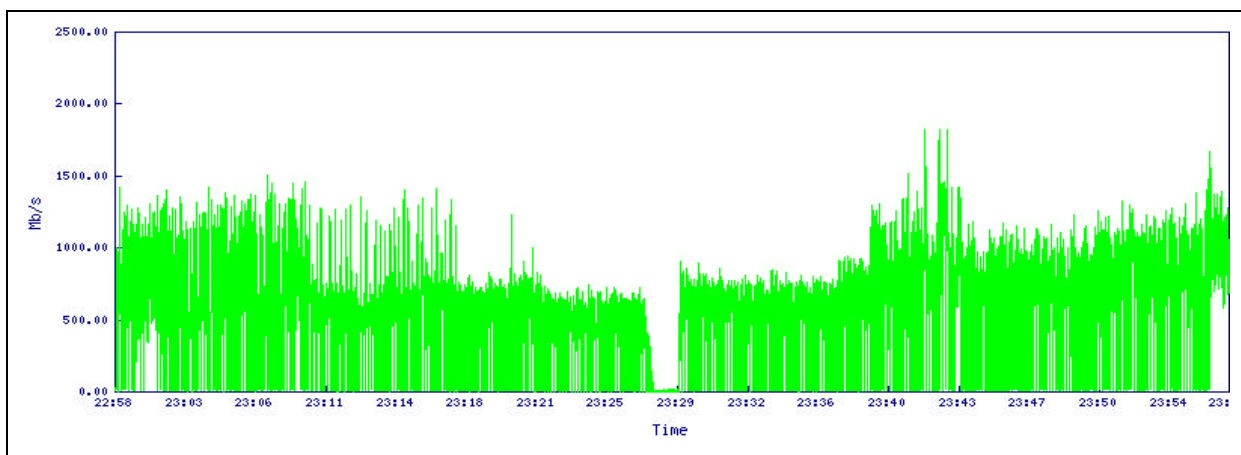


Figure 4: Throughput at SC 2000 router during Visapult Bandwidth Challenge run

## 4.2 GridFTP Results

The ANL entry in the network challenge was a climate modeling application. This application is representative of a wide range of data intensive applications. These applications require large amounts of data and, to reduce network overhead, often employ replication to make the data access more efficient and less costly. ANL, through the Globus project, provides the infrastructure required for such applications. During our hour of dedicated network access we were transferring data to create a new replica of climate data at LBNL. We were able to transfer 230.8 GB of data, for an aggregate data transfer rate of 512.9 Mb/s, with peaks of greater than 1 Gb/s over 5 second intervals.

On the show floor we had an eight node single CPU Linux cluster each equipped with 4 SCSI disks and a Gigabit Ethernet adapter. These were connected to a Cisco switch with dual bonded GigE out to the show floor routers, then via OC-48 to LBNL. The data files were 2 GB in size and were arranged using the partitioned layout on both source and destination. Each node used a parallelism of four (4 TCP streams for its share of the data), and there were as many as four files being transferred simultaneously. This resulted in a maximum of 128 total data streams (8 nodes x 4 streams x 4 files).

We performed 5-second polls to the LBNL router. A graph of the router throughput, smoothed with a 10-point averaging window to make trends in the data clearer, is shown below in Figure 5

The data transfer for this application was very uneven and “bursty” for two reasons. One is the nature of the transfer. A 2 GB file can be transferred in about 20 seconds and then the data connections must be torn down and a new transfer started. This causes spikes in the data transmission rate, especially when all four files ended at approximately the same time. We also had a several minute period where one of the receiving servers had crashed and we had to reboot, and restart the transfer.

## 5.0 Conclusions

The techniques described in this paper and implemented in both GridFTP and the DPSS will be needed to realize the potential of next generation high bandwidth networks. However, use of these techniques still requires extra effort and knowledge usually not available to the application programmer. We feel that the example implementations here show not only how to use these techniques, but also how these techniques can be accessed in a fashion that is not much different then that of a local standard file access, while at the same time taking full advantage of a high speed wide area network.

The basic functionality of GridFTP is currently in place. The code is in late alpha testing and should be going to beta soon. When released it will be available under the Globus public license at <http://www.globus.org>. As a result of our experiences at SC 2000 we have already made 2 small, but important improvements to our current implementation. We have added 64 bit file support for larger than 2 GB files, and we have added data channel caching. The data channel caching will be particularly useful since it will avoid the overhead of setup and tear down of the sockets, which can be significant, particularly when authentication is enabled on the data channels. We are also going to explore the possibility of implementing our striped server on top of a parallel virtual file system.

The DPSS project is now fully functional and can be downloaded from <http://www.didc.lbl.gov/DPSS>. While we are still making minor adjustments to the DPSS, we are mostly interested in looking at how high bandwidth data transfer needs can be integrated with higher-level services. We are investigating integration efforts in the areas of file

## GridFTP Throughput at LBNL Router

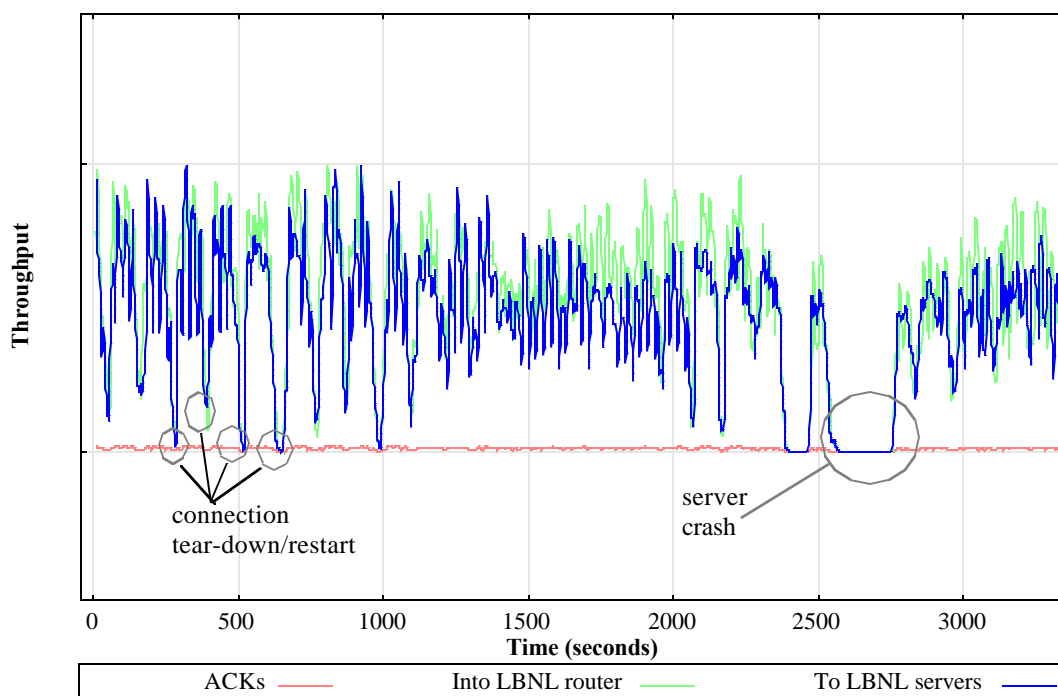


Figure 5: Throughput at LBNL router during GridFTP Bandwidth Challenge run, smoothed

replication, staging, caching, and location transparency. In addition, we are considering the use of dynamic data from performance monitoring as a feedback mechanism to a live transfer. We feel that monitoring data can contribute on several levels, such as what link to use, what storage resource to use in a replicated data set, and whether to move the data to the computation or vice-versa.

## 6.0 Acknowledgments

This work was supported by the Director, Office of Science. Office of Advanced Scientific Computing Research. Mathematical, Information, and Computational Sciences Division under U.S. Department of Energy Contract No. DE-AC03-76SF00098. This is report no. LBNL-47183.

## 7.0 References

- [1] B. Bershad et. al., “The Scalable I/O Initiative”, white paper, available through the Concurrent Supercomputing Consortium, CalTech, Pasadena, CA Feb. 1993, <http://www.cacr.caltech.edu/SIO/>
- [2] Bethel, et. al., “Bandwidth Statistics Reported by SciNet Router”, <http://www.didc.lbl.gov/presentations/SC00.LBNL.netchallenge.pdf>, Slide 5, November 2000
- [3] Bethel, W., Tierney, B., Lee, J., Gunter, D., Lau, S., “Using High-speed WANs and Network Data Caches to Enable Remote and Distributed Visualization”, Proceedings of the IEEE SuperComputing 2000 Conference, Nov. 2000, LBNL-45365
- [4] Carns, P., Ligon III, Ross, R., Thakur, R., “PVFS: A Parallel File System For Linux Clusters”, Proceedings of the 4th Annual Linux Showcase and Conference, Atlanta, GA, October 2000, pp. 317-327
- [5] The DataGrid Project: <http://www.cern.ch/grid/>
- [6] Dykstra, P., “Gigabit Ethernet Jumbo Frames”, <http://www.columbia.edu/acis/networks/advanced/jumbo/jumbo.html>
- [7] “Fibre Channel · Overview of the Technology”, Fibre Channel Industry Association (FCIA), <http://www.fibrechannel.com/technology/overview.html>
- [8] Globus: <http://www.globus.org>
- [9] Goland, Y. et. al, “HTTP Extensions for Distributed Authoring -- WEBDAV”, IETF RFC 2518, Feb. 1999



- [10] The Grid: Blueprint for a New Computing Infrastructure”, edited by Ian Foster and Carl Kesselman. Morgan Kaufmann, Pub. August 1998. ISBN 1-55860-475-8.
- [11] GridFTP: Universal Data Transfer for the Grid, White Paper, <http://www.globus.org/datagrid/deliverables/C2WPdraft3.pdf>
- [12] Hartman, J., Murdock, I., Spalink, T., “The Swarm Scalable Storage System”, Proceedings of the 19th IEEE International Conference on Distributed Computing Systems, June 1999.
- [13] Hartman, J., Ousterhout, J, “The Zebra Striped Network File System”, ACM Transactions on Computer Systems 13, 3, August 1995, 279-310.
- [14] Hoschek, W., J. Jaen-Martinez, A.Samar, H. Stockinger, K. Stockinger, “Data Management in International Data Grid Project”, to IEEE, ACM International Workshop on Grid Computing (Grid’2000), Bangalore, India, 17-20 Dec. 2000.
- [15] Hwang, K, J. Jin, P. Novaux, “RAID-x: A New Distributed Disk Array for I/O Centric Cluster Computing”, In Proc. 9th IEEE Symp. on High Performance Distributed Computing, Aug 2000
- [16] Jacobson, V., “Congestion Avoidance and Control,” Proceedings of ACM SIGCOMM ‘88, August 1988.
- [17] Kleinrock, L., Queueing Systems, Vols. I&II, J. Wiley and Sons, 1975.
- [18] Lakshman, T., Madhow, U., “The performance of TCP/IP networks with high bandwidth-delay products and random loss”, IEEE Transactions on Networking, vol. 5 no 3, pp. 336-350, July 1997
- [19] National Transparent Optical Network (NTON) <http://www.ntonc.org/>
- [20] Patterson, David A., G. Gibson, R. Katz, “A Case for Redundant Arrays fo Inexpensive Disks (RAID). In International Conference on Management of Data (SIGMOD), pages 109-116, June 1988
- [21] Postel, J. and Reynolds, J., “File Transfer Protocol (FTP)”, IETF RFC 959, October 1985
- [22] Semke, J. Mahdavi, M. Mathis, “Automatic TCP Buffer Tuning,” Computer Communication Review, ACM SIGCOMM, volume 28, number 4, Oct. 1998.
- [23] Sivakumar, H, S. Bailey, R. L. Grossman, “PSockets: The Case for Application-level Network Striping for Data Intensive Applications using High Speed Wide Area Networks”, Proceedings of IEEE Supercomputing 2000, Nov., 2000. <http://www.ncdm.uic.edu/html/psockets.html>
- [24] SuperNet Network Testbed Projects: <http://www.ngi-supernet.org/>
- [25] Shen, X, A. Choudhary, “A Distributed Multi-Storage Resource Architecture and I/O Performance Prediction for Scientific Computing”, In Proc 9th IEEE Symp. on High Performance Distributed Computing, Aug 2000
- [26] “TCP Tuning Guide”, <http://www.didc.lbl.gov/tcp-wan.html>
- [27] Transmission Control Protocol (TCP), IETF RFC 793, September 1981
- [28] Tierney, B., Johnston, W., Crowley, B., Hoo, G., Brooks, C., Gunter, D., “The NetLogger Methodology for High Performance Distributed Systems Performance Analysis”, Proceedings of the IEEE High Performance Distributed Computing conference (HPDC-7), July 1998, LBNL-42611
- [29] Tierney, B. J. Lee, B. Crowley, M. Holding, J. Hylton, F. Drake, “A Network-Aware Distributed Storage Cache for Data Intensive Environments”, Proceeding of IEEE High Performance Distributed Computing conference (HPDC-8), August 1999, LBNL-42896. <http://www.didc.lbl.gov/DPSS/>
- [30] Watson, R., Coyne, R., “The Parallel I/O Architecture of the High-Performance Storage System (HPSS)”, IEEE MS Symposium, 1995

# Enabling Network-Aware Applications

Brian L. Tierney, Dan Gunter, Jason Lee, Martin Stoufer

Computing Sciences Directorate  
Lawrence Berkeley National Laboratory  
University of California, Berkeley, CA, 94720

Joseph B. Evans  
Information & Telecommunication Technology Center  
University of Kansas, Lawrence, KS 66045

## Abstract

*Many high performance distributed applications use only a small fraction of their available bandwidth. A common cause of this problem is not a flaw in the application design, but rather improperly tuned network settings. Proper tuning techniques, such as setting the correct TCP buffers and using parallel streams, are well known in the networking community, but outside the networking community they are infrequently applied. In this paper, we describe a service that makes the task of network tuning trivial for application developers and users. Widespread use of this service should virtually eliminate a common stumbling block for high performance distributed applications.*

## 1.0 Introduction

Internet backbone speeds have increased considerably in the last few years due to projects like Internet II and NGI. At the same time, projects like NTON [25] and SuperNet [34] are providing a preview of the near future of wide area networks. Unfortunately, distributed applications often do not take full advantage of these new high-speed networks. This is largely due to the fact that the applications use the default parameters for TCP, which have been consciously designed to sacrifice optimal throughput in exchange for fair sharing of bandwidth on congested networks. In order to overcome this limitation, distributed applications running over high-speed wide-area networks need to become “network-aware” [32][36], which means that they need to adjust their networking parameters and resource demands to the current network conditions.

There exists a large body of work showing that good performance can be achieved using the proper tuning techniques. The most important technique is the use of the optimal TCP buffer size, and techniques for determining the optimal value for the TCP buffer size are described in [35]. Another important technique is to use parallel sockets, as described in [31]. Using a combination of these techniques, applications should be able to utilize all the available network bandwidth, which is demonstrated in [4], [1], and [16].

However, determining the correct tuning parameters can be quite difficult, especially for users or developers who are not network experts. The optimal TCP buffer size and number of parallel streams are different for every network path, vary over time, and vary depending on the configuration of the end hosts. There are several tools that help determine these values, such as *iperf* [14], *pchar* [26], *pipechar* [27], *netspec* [23], and *nettest* [22], but none of these include a client API, and all require some level of network expertise to use. Another tool is *NWS* [38], which applications can use to determine upper bounds on throughput from the network, but it does not tell the applications how to achieve that throughput. Other groups are addressing this problem at the kernel level, such as the web100 project [37], Linux 2.4 [17], and others [9], as described below. Still others are addressing this within the application. The *autoftp* file transfer service from NCSA [19] attempts to determine and set the optimal TCP buffer size for each connection.

In this paper we describe a service which provides clients with the correct tuning parameters for a given network path. We call this service **Enable**, because it enables applications to optimize their use of the network and achieve the highest possible throughput. The goal of the Enable service is to eliminate what has been called the “wizard gap” [21]. The wizard gap is the difference

---

This paper published in the proceedings of the Tenth IEEE International Symposium on High Performance Distributed Computing, August, 2001, San Francisco, CA.

between the network performance that a network “wizard” can achieve by doing the proper tuning, compared to the performance of an untuned application. The Enable service can act as that wizard. Enable hides the details of gathering the data from multiple network monitoring tools behind an intuitive, easy to use interface. From the application developer’s perspective, Enable provides advice on the correct tuning parameters without requiring knowledge about how these are obtained. Thus, the selected algorithms and tools for computing these parameters can be changed transparently to the application. This frees the distributed application developer from needing to understand the wide variety of available monitoring tools.

The Enable service works as follows: An Enable server is co-located on every system that is serving large data files to the wide-area network (e.g.: an FTP or HTTP server). The Enable service is then configured to monitor the network links to a set of client hosts from the perspective of that data server. Network monitoring results are stored in a database, and can be queried by network-aware distributed components at any time. The Enable service runs the network tests on some pre-configured time interval (e.g.: every 6 hours, or whenever a new client connects). The Enable service API makes it very easy for application or middleware developers to determine the optimal network parameters. To take advantage of the Enable tuning service, distributed applications must be modified to be support network tuning such as the ability to set the TCP buffer size [35] or the ability to create and use multiple data streams to transfer data in parallel.

The network tuning parameters that the Enable service is initially concentrating on are those required by large bulk data transfer applications, such as the various “Data Grid” [5] projects. These include the Particle Physics Data Grid [28], GriPhyn [10], the Earth Systems Grid [7], and the EU DataGrid [12]. These projects all require the efficient transfer of very large scientific data files across the network. We are not yet addressing the tuning requirements of other types of applications, such as latency-sensitive applications.

## 2.0 Background

TCP uses what it calls the “congestion window” to determine how many packets can be sent at one time. The larger the congestion window size, the higher the throughput. The TCP “slow start” and “congestion avoidance” algorithms determine the size of the congestion window [20]. The maximum congestion window is related to the amount of buffer space that the kernel allocates for each socket. For each socket, there is a default value for the buffer size, which can be changed by the program using a system library call just before opening the socket.

The buffer size must be adjusted for both the send and receive ends of the socket. To get maximal throughput it is critical to use optimal TCP send and receive socket buffer sizes for the link you are using. If the buffers are too small, the TCP congestion window will never fully open up. If the buffers are too large, the sender can overrun the receiver, and the TCP window will shut down. The optimal TCP window size is the bandwidth delay product for the link. For more information, see section 5, and [30] and [36].

As network throughput speeds have increased in recent years, operating systems have gradually changed the default buffer size from common values of 8 kilobytes to as much as 64 kilobytes. However, this is still far too small for today’s high speed networks.

For example, there are several hosts which are part of the Particle Physics Data Grid [28] with 1000 BT network interfaces which are connected via an OC12 (622 Mbit/sec) WAN, with typical round-trip network latencies of about 50 ms. For this type of network, the bandwidth delay product, and hence the TCP buffer, should be roughly 3.75 MBytes. Using a default TCP buffer of 64 KB, the maximum utilization of the pipe will only be about 2% under ideal conditions. Furthermore, 10 Gbit/sec ethernet and OC192 WAN’s (9.6 Gbit/sec) are just becoming available, which will require TCP buffer sizes of roughly 62 MBytes per connection to fully utilize the link! (However, typical workstations today can, at best, drive the network at about 1 Gbit/sec, so TCP buffers requirements of this size are still a couple of years away)

As the awareness of the importance of TCP buffer tuning has increased, several data transfer tools now include the ability for the user to set this value. For example the *gsiftp* [1][11], *bbftp* [3], *SRB* [2], *HPSS* [13], and *DPSS* [36] all provide this ability. Additionally, some systems, such as DPSS and *gsiftp*, also support the ability for users to request parallel data streams. The *psockets* library from the University of Illinois makes it easy for applications developers to add parallel sockets to their applications [31].

Figure 1 shows the advantage of using tuned TCP buffers and parallel streams in the *gsiftp* program for 100 MByte data transfers between Lawrence Berkeley National Lab in Berkeley, CA, and CERN in Geneva, Switzerland. The round trip time (RTT) on the connection was measured with *ping* to be 180 ms and the bottleneck link was measured with *pipechar* to be 45 Mbit/sec. With different tuning parameters, actual measured transfer speeds spanned more than an order of magnitude. Tuned TCP buffers alone provided a 9x performance increase, and parallel sockets alone yielded a 12x performance improvement. Using parallel streams with tuned TCP buffers we were able to saturate the network. This combination of techniques provided a 15x performance increase, which was an

additional 40% improvement over just tuned buffers and a 26% improvement over just parallel streams.

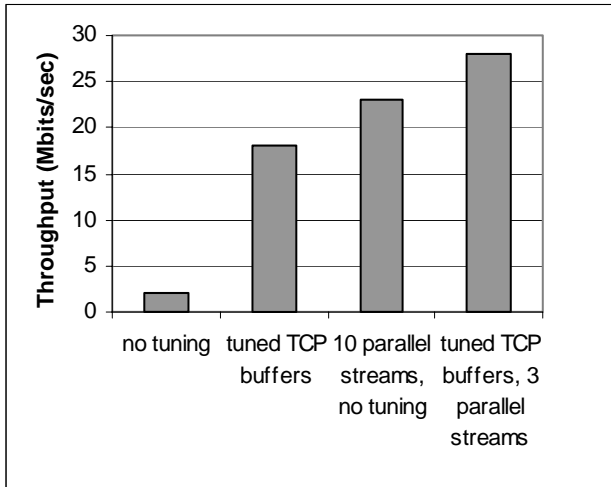


Figure 1: gsiftp results using tuned TCP buffers and parallel streams

The use of parallel streams provides an increase over optimally tuned single stream TCP because TCP is rather sensitive to any loss or congestion, and slows down whenever any loss is detected. Our testing has shown that it is extremely rare that a TCP stream keeps its congestion window at the optimal bandwidth delay product size for very long. The use of multiple streams allows one to utilize a greater fraction of the network. Note that this may be considered a “rude” thing to do, depending on how congested the network is and whether or not you are slowing down others by doing this.

However, as with all systems that provide the ability to tune the TCP buffer size or the number of parallel streams, the *gsiftp* user must set these values by hand, and determining what values to use is not simple. In general, using large TCP buffers and parallel streams improves throughput, so it may be tempting for users or developers to simply use big buffers and some parallel streams by default. However this is not a good idea. Besides wasting operating system resources, under certain circumstances overly large TCP buffers or too many parallel streams can significantly decrease performance, as shown in the Tables 1 and 2.

Table 1 shows the result of tests between two 333 MHz Sun Ultra 1 hosts running Solaris 2.7, connected by a Gigabit Ethernet LAN with a 1500 byte MTU (maximum transmission unit). Note that setting the TCP buffer too large results in a large performance loss. This is because when the buffers are too large, the sender can overrun the receiver, and the TCP window will shut down. Not all operating systems have this behavior (e.g.: Linux does not), but this reemphasizes that taking the simple approach of just setting large buffers everywhere is not a good idea.

Table 2 shows the result of tests between a Sun Ultra 1 (333 Mhz) sender and a 450 MHz Pentium II Linux 2.2 receiver also over a OC-12 WAN with a 1500 byte MTU. In this case we see a large performance penalty using parallel data streams. This is because a 450 MHz PII processor is not powerful enough to handle load from the Gigabit network interface card. It requires most of the CPU just to read one stream, and multiple streams just step on each other.

The Enable service makes it easy for applications to use the correct settings and avoid these types of problems.

Table 1 : Sender overruns receiver

TCP Buffer Size (MB)	Throughput (Mbits/sec)
0.125	246
1	195
4	105
8	32

Table 2 : Parallel streams

Number of Streams	Total Throughput (Mbits/sec)
1	250
2	100
4	50

### 3.0 Related Work

There are a number of tools to help determine the optimal TCP parameters for a given network path. For example, one can run a series of *iperf* tests with a range of buffer sizes and numbers of parallel data streams to determine the optimal values. Other tools such as *pchar* [26], *pipechar* [27], and *pathrate* [6] can be used to estimate the bandwidth and latency characteristics of the network, providing information needed to estimate the optimal TCP buffer size. However, these tools do not include a client API, and require some level of network expertise to use. The Enable service can be used to run any of these tools, collect and store the results, and make the results available to network-aware applications.

Additionally, there are some other projects that are also working on eliminating the “wizard gap”. The *web100* project is developing a version of the Linux kernel which will perform dynamic, transparent, automatic TCP tuning for user level processes. If successful, this has the potential to eliminate the TCP buffer tuning issue. Fisk and Feng [9], have also demonstrated promising results with Linux kernel

modifications that autotune the TCP buffer size by estimating link bottleneck bandwidth for each socket connection.

The Linux 2.4 kernel also includes an option for TCP buffer autotuning, and initial testing shows that this helps quite a bit, but is still not as good as hand tuning (see the results section below). Unfortunately the developers of this code are not part of the IETF or any TCP research community, and any solution they come up with is not likely to be standardized or adopted very quickly.

Therefore, while there is some hope that automatic TCP buffer tuning will be built into some operating systems in the future, it will probably not be built into most operating systems in the near future.

## 4.0 The Enable Service

The Enable service has three distinct components. First, there is the Enable Server, which keeps an up-to-date record of network parameters between itself and other hosts. The second component is a protocol for clients to communicate with the servers. Finally, there is a simple API that makes querying the Enable Servers trivial for application developers. A primary design goal for the Enable service was ease of installation, configuration, and use.

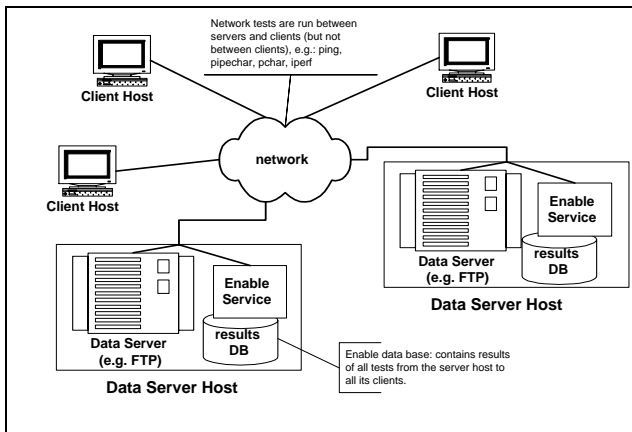


Figure 2: Enable Service Architecture

The architecture of Enable is shown in Figure 2. The simplicity of the design is its strength. An Enable Server is installed on every data server host, such as an *FTP* server, and that Enable server is responsible only for determining the optimal network parameters between clients and itself. Other monitoring systems, such as *NWS*, can be configured to monitor an arbitrary mesh, or “clique” of hosts. This design, while very powerful, makes these systems more complicated to deploy and configure, as it requires software to be installed on every host in the clique. We have decided to sacrifice this functionality for ease of deployment and configuration. In return, we avoid the problems of

centralized coordination and location of the Enable servers, as they are always co-located with the data server.

The following section describes the functionality and implementation of the Enable Service.

### 4.1 Functionality

The Enable Server will periodically run tests between itself and a number of “client hosts”. These client hosts may have been read at start-up from a configuration file, manually added using an API or command-line utility, or automatically added by monitoring log files from the data server, such as *HTTP* or *FTP* logs. The results of the network tests will be stored in a database. The selection and scheduling of tests for each client is dynamically configurable.

Clients can query the Enable server, which is listening on a well-known port, for network parameters, also called “network advice”. The protocol for doing this is *XML-RPC* [39], a standard XML-based protocol that performs remote procedure calls over *HTTP*. Use of a standard protocol means that third parties can easily interface with Enable without using the Enable API or libraries.

There is a simple API that clients can use to query the Enable Server. For example:

```
tcp_buffer_size =
    EnableGetBufferSize(ftp_hostname)
```

returns the optimal buffer size between itself and the *FTP* server host, and:

```
net_info =
    EnableGetNetInfo(ftp_hostname)
```

returns the result of all network tests for that network path. One could also wrap an application in a script that called the Enable Server, and then set the buffer size via a command line argument. For example, we have written a script that automatically finds and sets the “-B” flag (which sets the TCP receive buffer) for the *ncftpget* *FTP* client program [24].

Currently the Enable server supported network tests are *ping*, *pipechar*, *pchar*, and *iperf*, but only *ping* and *pipechar* are run by default.

Since the network tests are run periodically, there is the possibility that one of the tests will be run during some unusual network problem, and the results of this test will not lead to useful results for tuning applications. Therefore, a trimmed mean, in which the top and bottom 10% of values are discarded before calculating the mean of the most recent *N* values (*N* is configurable, default is 10), is reported to the client.

In order to more quickly detect a long-term shift in network behavior, the mean and standard deviation of the last *N* values if also calculated. If three successive values are farther than 2.5 standard deviations from the mean, it is

assumed that the network behavior has changed, and the older N-3 values are discarded. This approach is based on the assumption that the distribution of test results closely approximates a normal distribution. More testing is needed to validate this method for handling data fluctuations.

## 4.2 Use-case

In this section we illustrate the use of the Enable service with a simple use-case in a Data Grid application. In the EU DataGrid project [8], huge volumes of high-energy physics data must be replicated at several sites around the world. For example, five sites may wish to create a replica of a particular set of data that is stored on a data server at CERN in Geneva, Switzerland. In this project, *gsiftp*, a data transfer utility based on FTP that provides TCP buffer tuning and parallel stream support is used to transfer data between CERN and each of the other sites.

In this environment, there is a large variability in delay and bandwidth to each of the replication sites, as shown in Figure 3. Note that no statically configured TCP buffer size will work well for all the clients: a buffer of 256 KBytes will penalize clients A, B and E while a buffer of 1-2 MBytes will penalize A, C, and D (due to effects shown in Table 1). Data Grid file transfer tools such as *gsiftp* allow the users to specify a buffer size. However this solution is far from optimal, as it requires too much knowledge and work on the part of the users. Instead, the *gsiftp* client can be wrapped in a script that uses the Enable service find the optimal TCP buffer size for each path.

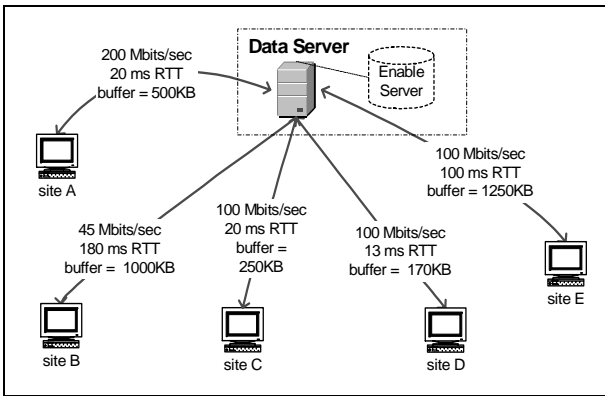


Figure 3: Data Grid Use Case

## 4.3 Implementation

The Enable server is implemented using the Python language [29], and uses XML-RPC [39] for client-server communication. The use of Python with XML-RPC greatly simplified the development of the server, as Python includes very powerful built in modules for threads, queues, databases, regular expressions, configuration file parsing: i.e. almost everything required by this service. XML-RPC

was chosen rather than SOAP [33] because the current SOAP implementations are still evolving, and because XML-RPC is simpler and provides everything we need.

The server uses a thread pool of worker threads for running the network tests, and a scheduler thread to feed jobs to the workers. By limiting the number of worker threads it is easy to limit the amount of load generated by the testing. There is also a thread for scanning log files (e.g. FTP logs) for new hosts to monitor. We have developed client APIs for the Python, Java, and C languages.

Enable was designed for the easy addition of new tests, and each test is realized by a class instance in Enable. Enable requires only 3 specific methods in the new class to be implemented: “init”, “can\_I\_run” (is it safe to start this test), and “run”.

We have tested an Enable server that was configured to monitor 500 hosts, running each test every 4 hours using 8 worker threads, on a 500 MHz PIII Linux host. While running tests, the Enable server consumed at most 9% of the CPU, and used an average of only 130 Kbits/sec of network bandwidth. (By default, 10 ping tests are run in parallel, and use 12 Kbits/sec each, and only 1 pipechar can run at a time, which generates only about 100 Kbits/sec of network traffic). There are still some scalability issues to address, as discussed in the section on future work below.

## 5.0 Results

To test the results of the Enable service, we used *iperf* as a client/server pair over four different network paths: LBL (Berkeley, CA) to CERN (Geneva, Switzerland); LBL to ISI (Arlington, VA) over SuperNet; LBL to the University of Kansas (Lawrence, KS), and ANL (Chicago, IL) to SRI in Menlo Park, CA. Characteristics of these network paths are summarized in Table 3. *iperf* was chosen for testing because it is a simple tool that only performs network transfers, thus ensuring that we are only measuring network performance, and not some combination of network, disk, and application performance.

Table 3 Test network path characteristics.

Path	Round Trip Time (RTT)	Bottleneck Link Bandwidth
LBL-CERN	180 ms	45 Mbits/sec
LBL-ISI East	80 ms	1000 Mbits/sec
LBL-ANL	60 ms	45 Mbits/sec
LBL-KU	50 ms	45 Mbits/sec

The results are shown in Table 4. All testing used Linux 2.4 as a sending host. The first row is the results with no tuning (with the default TCP buffers set to 64 KBytes, and Linux 2.4 autotuning disabled). The second row shows results for the Linux 2.4 autotuning option, with autotuning

Table 4 . Experimental throughput using four tuning methods.

Tuning Method	LBL-CERN	LBL-ISI east	LBL-ANL	LBL-KU
No Tuning	2 Mbits/sec	5 Mbits/sec	5 Mbits/sec	6 Mbits/sec
Linux 2.4 Autotuning	6 Mbits/sec	110 Mbits/sec	12 Mbits/sec	9 Mbits/sec
Hand Tuning	18 Mbits/sec	266 Mbits/sec	17 Mbits/sec	27 Mbits/sec
Enable Tuning	18 Mbits/sec	264 Mbits/sec	16 Mbits/sec	26 Mbits/sec

parameters set to allow up to 4 MByte TCP buffers. The third row is hand-tuned *iperf*, meaning that *iperf* was run with a range of TCP buffer settings, and the setting which gave the maximum throughput is shown here. The fourth row is the result from *iperf* using the TCP buffer size value returned by the Enable service, which used *ping* and *pipechar* to estimate the optimal TCP buffer size using the following standard formula, as described in [35]:

$$\text{optimal TCP buffer} = \text{RTT} \times (\text{speed of bottleneck link})$$

The Enable server runs a *ping* test, sending a 1500 byte packet 5 times. The round trip time is estimated to be the average time for ping packets 2-5. The Enable server also runs *pipechar* with the *-bot* option, which gives the speed of the bottleneck hop in the network path between the Enable server host and the client.

From this table one can see that Linux 2.4 autotuning helps considerably, but not as much as hand tuning and Enable tuning. Hand-tuned and Enable-tuned clients both had nearly identical results. Note that when doing this type of testing on production networks, the variability of the results is very high, and these numbers are all just rough estimates. However, the overall improvements from tuning are quite clear.

## 6.0 Scalability Issues

We are currently addressing a number of scalability issues that arise when running active network test tools.

### 6.1 Aggregation for Measurement Efficiency

In order to scale the Enable service to networks with many clients, measurements need to be aggregated to avoid redundant tests for hosts on the same subnet. Aggregation involves the abstraction of a set of individual pairwise performance behaviors by a single performance characteristic. This is a widely used method to improve the scalability of routing and quality of service schemes. Unfortunately, there is a fundamental trade-off between precision and scalability in any such aggregation technique. The Enable service is implementing several schemes, discussed below, which may be selected based on the preferred policy.

The default, and likely most precise, approach is to measure each pairwise path with a reasonably high rate of

repetition. The approaches that follow attempt to improve efficiency while maintaining a reasonable level of precision.

A fairly conservative policy is to measure all clients at least once to insure precision. This approach allows a reasonably reliable database of paths and bottlenecks to be developed. By measuring the pairwise behavior at least once, some network pathologies can be avoided. For example, two clients might appear to be on the same subnet, but one might be directly connected via Ethernet, while the other is connected via a (relatively slow) dialup server. The bottleneck in the former case would likely be somewhere in the wide area network, while the dialup link would be the constraint in the latter case. Direct measurement would clearly identify the differing bottleneck locations.

Once the performance of a client/server pair is measured and a bottleneck link is identified, a table of clients and bottlenecks can be created. The Enable service then suppresses additional redundant testing to clients with the same bottleneck link, and sets a time after which further pairwise testing might be performed. Tests to one of the clients behind the bottleneck can still be performed more frequently to update the state of the constraining link.

An example appears in Table 5. In this example, it can be seen that the bottleneck for clients 129.237.116.6 and 129.237.127.152 is the same, that is, 164.113.232.202. Occasional testing to one of 129.237.116.6 or 129.237.127.152, but not both, would be performed to update the state of the performance constraint.

Table 5 Bottlenecks to Clients

Client	Bottleneck Router
129.237.116.6	164.113.232.202
129.237.127.152	164.113.232.202
131.243.2.12	131.243.128.100
131.243.2.91	131.243.128.100
192.195.6.68	144.232.0.171

The Enable service also implements more aggressive, less precise schemes for aggregation of measurements.

A simple approach is to base the decision on the bottleneck characteristics. Tools such as *pipechar* provide

both bottleneck identification and *traceroute* information from server to client. If *pipechar* indicates that a host is behind a known bottleneck with particular characteristics (perhaps below a certain bandwidth threshold), any subsequent clients appearing behind that bottleneck might receive like treatment. For example, if the bottleneck is below T1 rates, it might be assumed that all other clients behind that bottleneck, as determined by *traceroute*, are limited by that particular link and that no additional tests are necessary within a certain time frame.

Another scheme is based upon identification of subnets. In particular, a client sharing an IP address prefix with another client already in the table gets similar treatment. The extent of the client network might be based on routing advertisements, and determined by querying a Looking Glass [18] server. This obviously abstracts away the internal details of the client networks in favor of simplicity.

The choice of aggregation policies can be determined when the service is configured.

## 6.2 Measurement Frequency

Sophisticated mechanisms for controlling the test frequency are also needed to provide scalability.

The Enable service can base these decisions on the measurements themselves and on the client requests. In particular, the measurements on a particular path will likely be correlated in time. The degree of time correlation can be used to determine the valid period for a particular measurement, and hence the time at which testing should be resumed. This can also be combined with the client requests (specifically the size of transfer requested) to determine if additional measurements need to be derived from the transfer itself and the parameters need to be updated accordingly.

In addition, it is necessary that the service implement an aging and purging mechanism to remove old clients so that the database size does not increase monotonically.

## 6.3 Other Scaling Issues

There are other ways in which scalability can be improved. For example, the Enable service should have the ability to monitor the load that all its tests are placing on the network, to ensure that its total load does not exceed some predefined threshold. The Enable architecture allows a single server to implement this in a straightforward fashion. Future work might investigate ways in which Enable servers on the same network might coordinate to control testing loads on shared paths.

## 7.0 Future Work

A great deal of work remains to be done on the Enable service. The next scheduled addition is the ability to give advice on the number of parallel streams to use. Our tests

have shown that the optimal number of streams depends on a number of factors, including host load / processing power, and congestion of the network. The Enable server will base its estimate on both the client library's estimate of the host CPU speed and the server's network testing results.

We also plan to do more detailed analysis of the results of the various network tests, so that we can detect anomalies and make better TCP window estimates. When we can accurately identify results that lie outside the realm of normal measurement error, we might throw out the value, flag the result as a "temporary anomaly", generate an email message to a network administrator, and so on.

Another issue we need to address is that of asymmetric paths. Internet routing data has shown that as many as 20% of the paths are asymmetric, especially very long paths [15]. Any measurements or tuning based on round-trip time on an asynchronous path may be meaningless. We want to explore this issue further.

The other future work that we have planned is to add support for providing network Quality of Service (QoS) advice. There are many predictions that soon networks will support various levels of QoS, and applications will be able to request a given QoS level depending on application requirements. We envision that the decision of which QoS level to request will be even more difficult than determining the optimal TCP buffer setting, and we believe the Enable service has the potential to help applications with this decision.

## 8.0 Conclusions

Network tuning is critical for applications to fully utilize high-speed networks, yet determining the proper tuning parameters can be quite difficult, especially for users who are not network "wizards". The Enable service described here can help applications achieve the same performance as hand-tuned applications. We believe the most valuable use of the Enable Service will be in Data Grid applications, where by installing an Enable Server on each Data Grid file server, applications can easily maximize their throughput to or from those servers.

The Enable server and client libraries are available for download at <http://www-didc.lbl.gov/ENABLE/>.

## 9.0 Acknowledgments

This work was supported by the Director, Office of Science. Office of Advanced Scientific Computing Research. Mathematical, Information, and Computational Sciences Division under U.S. Department of Energy. The LBNL work is under Contract No. DE-AC03-76SF00098, and the University of Kansas work is under Contract No. DE-FC03-99ER25399. This is report no. LBNL-47611.



## 10.0 References

- [1] Allcock B., Bester, J., Bresnahan, J., Chervenak, A., Foster, I., Kesselman, C., Meder, S., Nefedova, V., Quesnel, D., Tuecke, S., "Secure, Efficient Data Transport and Replica Management for High-Performance Data-Intensive Computing", <http://www.globus.org/>
- [2] Baru, C., R. Moore, A. Rajasekar, M. Wan, "The SDSC Storage Resource Broker," Proc. CASCON'98 Conference, Nov.30-Dec.3, 1998, Toronto, Canada.
- [3] bbfftp: <http://ccweb.in2p3.fr/bbfftp/>
- [4] Bethel, W., B. Tierney, J. Lee, D. Gunter, S. Lau, "Using High-Speed WANs and Network Data Caches to Enable Remote and Distributed Visualization", Proceeding of the IEEE Supercomputing 2000 Conference, Nov. 2000.
- [5] Chervenak, A., Foster, I., Kesselman, C., Salisbury, C. and Tuecke, S. "The Data Grid: Towards an Architecture for the Distributed Management and Analysis of Large Scientific Data Sets". Journal of Network and Computer Applications, 2000.
- [6] Dovrolis, C., Ramanathan P., Moore. D., "What Do Packet-Dispersion Techniques Measure?", Proceedings of the 2001 Infocom, Anchorage AK, April 2001.
- [7] Earth Systems Grid Project: <http://www.scd.ucar.edu/css/esg/>
- [8] EU DataGrid Project, <http://www.eu-datagrid.org/>
- [9] Fisk, M., Feng, W., "Dynamic Adjustment of TCP Window Sizes", LANL Report: LA-UR 00-3221.
- [10] GriPhyN Project: <http://www.griphyn.org/>
- [11] "GridFTP: Universal Data Transfer for the Grid", White Paper, <http://www.globus.org/datagrid/>
- [12] Hoschek, W., Jaen-Martinez, J., Samar, A., Stockinger, H. and Stockinger, K., Data Management in an International Data Grid Project. In Proc. 1st IEEE/ACM International Workshop on Grid Computing, 2000, Springer Verlag Press. Bangalore, India, December 2000. <http://www.cern.ch/grid/>
- [13] HPSS, "Basics of the High Performance Storage System", <http://www.sdsc.edu/projects/HPSS/>
- [14] iperf: <http://dast.nlanr.net/Projects/Iperf/index.html>
- [15] Kalidindi, S., Zekauskas, M., "Surveyor: An Infrastructure for Internet Performance Measurements", Proceedings of INET '99, [http://www.isoc.org/inet99/4h/4h\\_2.htm](http://www.isoc.org/inet99/4h/4h_2.htm)
- [16] Lee, J., D. Gunter, B. Tierney, W. Allock, J. Bester, J. Bresnahan, S. Tuecke, "Applied Techniques for High Bandwidth Data Transfers across Wide Area Networks", Dec. 2000, <http://www.didc.lbl.gov/publications.html>
- [17] Linux 2.4 autotuning: <http://www.linuxhq.com/kernel/v2.4/doc/networking/ip-sysctl.txt.html>
- [18] Looking Glass: <http://www.traceroute.org/>
- [19] Lui J., and Ferguson, J., "Automatic TCP socket buffer tuning", in Supercomputing 2000 Research Gems, Nov. 2000, <http://dast.nlanr.net/Features/Autobuf/>
- [20] Jacobson, V., "Congestion Avoidance and Control," Proceedings of ACM SIGCOMM '88, August 1988.
- [21] Mathis, M., "Pushing Up Performance for Everyone", Talk Slides, [http://www.ncne.nlanr.net/news/workshop/1999/991205/Talks/mathis\\_991205\\_Pushing\\_Up\\_Performance/](http://www.ncne.nlanr.net/news/workshop/1999/991205/Talks/mathis_991205_Pushing_Up_Performance/)
- [22] Netteest: "Secure Network Testing and Monitoring", <http://www-itg.lbl.gov/nettest/>
- [23] "NetSpec: A Tool for Network Experimentation and Measurement", Information & Telecommunication Technology Center, University of Kansas, <http://www.ittc.ukans.edu/netspec/>
- [24] NCFTP: <http://www.ncftp.org/>
- [25] National Transparent Optical Network (NTON); <http://www.ntonc.org/>
- [26] pchar: <http://www.employees.org/~bmah/Software/pchar/>
- [27] Jin, G., Yang, G., Crowley, B., Agarwal, D., "Network Characterization Service", Proceedings of the IEEE High Performance Distributed Computing conference, August 2001, <http://www-didc.lbl.gov/NCS/>
- [28] Particle Physics Data Grid: <http://www.ppdg.org/>
- [29] python: <http://www.python.org/>
- [30] Semke, J. Mahdavi, M. Mathis, "Automatic TCP Buffer Tuning," Computer Communication Review, ACM SIGCOMM, volume 28, number 4, Oct. 1998.
- [31] Sivakumar, H, S. Bailey, R. L. Grossman, "PSockets: The Case for Application-level Network Striping for Data Intensive Applications using High Speed Wide Area Networks", Proceedings of IEEE Supercomputing 2000, Nov., 2000. <http://www.ncdm.uic.edu/html/psockets.html>
- [32] Steenkiste, P., "Adaptation Models for Network-Aware Distributed Computations," 3rd Workshop on Communication, Architecture, and Applications for Network-based Parallel Computing, Orlando, January, 1999.
- [33] <http://www.w3.org/TR/SOAP/>
- [34] SuperNet Network Testbed Projects: <http://www.ngi-super-net.org/>
- [35] Tierney, B. "TCP Tuning Guide for Distributed Application on Wide Area Networks", Usenix ;login, Feb. 2001 (<http://www-didc.lbl.gov/tcp-wan.html>).
- [36] Tierney, B. Lee, J., Crowley, B., Holding, M., Hylton, J., Drake, F., "A Network-Aware Distributed Storage Cache for Data Intensive Environments", Proceeding of IEEE High Performance Distributed Computing conference (HPDC-8), August 1999, LBNL-42896.
- [37] "The WEB100 Project, Facilitating Effective and Transparent Network Use", <http://www.web100.org/>
- [38] Wolski, R., N. Spring, J. Hayes, "The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing," Future Generation Computing Systems, 1999. <http://nws.npaci.edu/NWS/>.
- [39] XML-RPC: <http://www.xmlrpc.org/>

# Using High-Speed WANs and Network Data Caches to Enable Remote and Distributed Visualization

Wes Bethel<sup>1</sup>, Brian Tierney<sup>2</sup>, Jason Lee<sup>2</sup>, Dan Gunter<sup>2</sup>, Stephen Lau<sup>1</sup>

*Lawrence Berkeley National Laboratory*

*University of California, Berkeley*

*Berkeley, CA 94720*

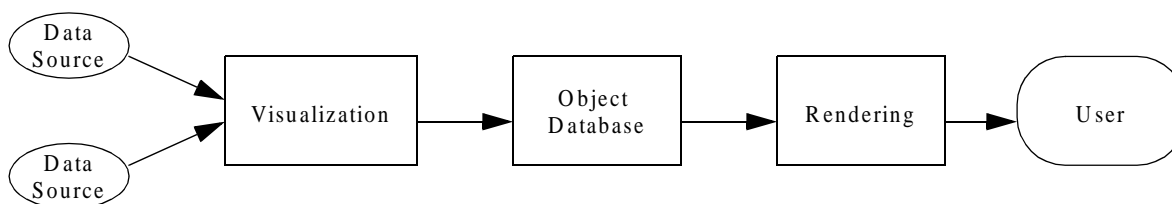
## 1.0 Abstract

Visapult is a prototype application and framework for remote visualization of large scientific datasets. We approach the technical challenges of tera-scale visualization with a unique architecture that employs high speed WANs and network data caches for data staging and transmission. This architecture allows for the use of available cache and compute resources at arbitrary locations on the network. High data throughput rates and network utilization are achieved by parallelizing I/O at each stage in the application, and by pipelining the visualization process. On the desktop, the graphics interactivity is effectively decoupled from the latency inherent in network applications. We present a detailed performance analysis of the application, and improvements resulting from field-test analysis conducted as part of the DOE Combustion Corridor project.

## 2.0 Introduction

As computing power increases, scientific simulations and instruments grow in size and complexity, resulting in a corresponding increase in output. During recent years, the increases in speed of infrastructure components that must absorb this output, including storage systems, networks and visualization engines, has not paced the increases in processor speeds. In response, solutions have tended toward parallel aggregations of slower, serial components, such as file systems striped across disk units.

**FIGURE 1. Visualization and Rendering Pipeline**



In particular, visualization and rendering pose interesting challenges as data sizes increase. In the visualization and rendering pipeline (Figure 1), abstract scientific data is first transformed into renderable data, such as geometry and image-based data, through the process of visualization. The resultant, renderable data is then transformed into a viewable image by a “draw” or rendering process. The challenges posed by large-model visualization stem from the sheer size of the data; it often won’t fit within the confines of primary or secondary storage on a typical desktop workstation. Movement of large amounts of data to the

1. [ewbethel, slau]@lbl.gov, Visualization Group.

2. [bltierney, jrlee, dkgunter]@lbl.gov, Distributed and Data Intensive Computing Group.

workstation over typical network links is impractically slow, but even if practical, the graphics systems of even high-end workstations quickly become overwhelmed.

Traditionally, visualization of large models has been approached using one of two strategies. In the first strategy, which we'll call "render remote," images are created on a large machine, preferably the same machine that has direct access to the data source (local filesystem), then transmitted to the user who views them on a workstation. In Figure 1, the link between *Rendering* and *User* would be over a network connection. In this configuration, a high-capacity resource has the potential to be applied to larger-sized problems than could be addressed with desktop resources, but graphics interactivity suffers due to the combination of latency and high bandwidth requirements<sup>3</sup>. In the second strategy, which we'll call "render local," smaller portions of the data, subsets or decimated versions of the raw data, are sent to the workstation where visualization and rendering take place. The network connection in this case is between *Data Source* and *Visualization*. Increasing graphics capacity mitigates concerns about interactivity, but the constraints encountered when moving remote data to the local workstation are exacerbated by limited network bandwidth and local storage capacity.

In recent years, two key developments have motivated us to explore a slightly different approach. One development is a network data cache that is tuned for wide-area network access, called the Distributed Parallel Storage System [1], or DPSS. The DPSS is a scalable, high-performance, distributed-parallel data storage system developed at Lawrence Berkeley National Laboratory (LBL). The DPSS is a data block server, built using low-cost commodity hardware components and custom software to provide parallelism at the disk, server, and network level. This technology has been quite successful in providing an economical, high-performance, widely distributed, and highly scalable architecture for caching large amounts of data that may potentially be used by many different users. Current performance results are 980 Mbps across a LAN and 570 Mbps across a WAN.

The other key development is a proliferation of high-speed, testbed networks. There are currently a number of Next Generation Internet networks whose goal is to provide network speeds of 100 or more times the current speed of the Internet. These include NSF's Abilene [2], DARPA's Supernet [3], and the ESnet testbeds [4]. Sites connected to these networks typically have WAN connection at speeds of OC12 (622 Mbps) or OC48 (2.4 Gbps); speeds that are greater than most local area networks (LANs). Access to these networks enables new options for remote, distributed visualization.

The combined capabilities of emerging high speed networks and scalable network storage makes it possible to consider remote, distributed scientific visualization from a new perspective, one which combines the best of both traditional methods.

### 3.0 Visapult: A Remote, Distributed Visualization Application Prototype

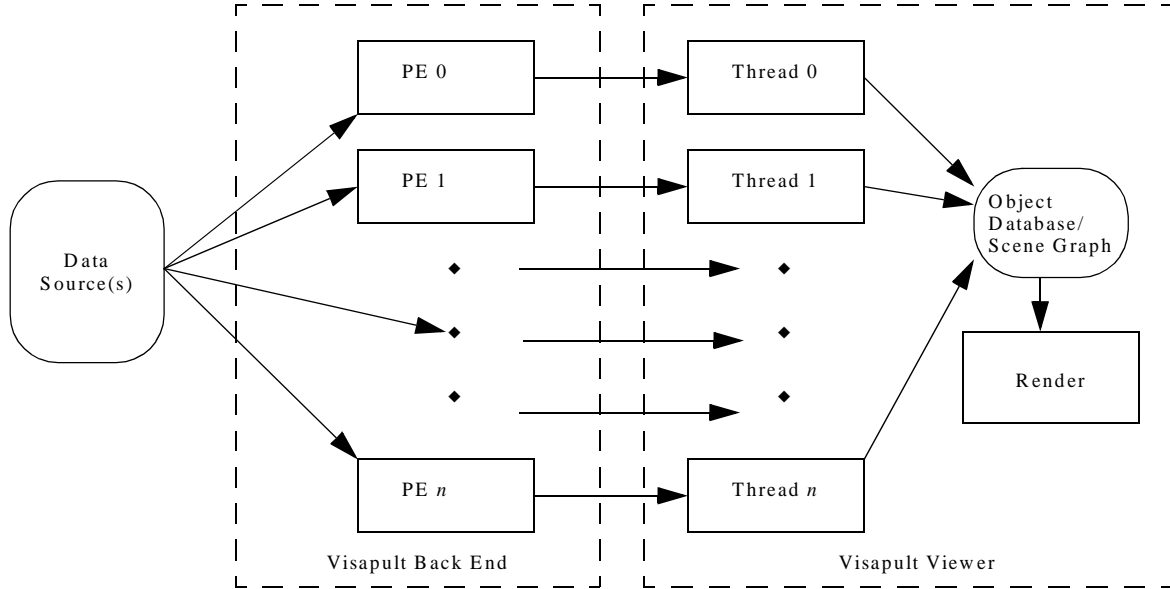
The Visapult application and framework consists of two distributed components (Figure 2): a viewer and a back end. In the following sections, we discuss the architecture of these components. The rendering portion of the viewer is built upon a scene graph model that proves useful for both asynchronous updates, as well as acting as a framework for the display of divergent types of data. The back end is a parallelized software volume rendering engine that uses a domain-decomposed partitioning, including the capability to perform parallel read operations over the network to a storage cache as well as parallel I/O to the viewer. Together, the viewer and back end implement a novel form of volume visualization that is fast but effective. More importantly, this novel form of volume visualization has been completely parallelized through

---

3. 1K by 1K, RGBA images at 30fps requires a sustained transfer rate of 960Mbps.

the visualization and rendering pipeline, from the data source to the display. We describe our use of the DPSS as a network storage cache, as well as our methodology for obtaining performance data from the application.

**FIGURE 2. Visapult Architecture**



### 3.1 Visualization and Rendering Pipeline Architecture

The fundamental goal of Visapult, from a visualization perspective, is to provide the means to visualize and render large scientific data sets with interactive frame rates on the desktop or in an immersive virtual reality (VR) environment. In our design, we wanted the best of both worlds: performing as much visualization and rendering as possible on a parallel machine with either tera-scale data storage capacity, or a high-speed network link to such a storage resource, while leveraging the increasing graphics capacity of desktop and desktide workstations. A primary Visapult design goal, graphics interactivity, is a crucial, but subtle, part of the visualization process; studies have shown that motion parallax and a stereo display format increase cognitive understanding of three dimensional depth relationships by 200%, as compared to viewing the same data in a still image [7].

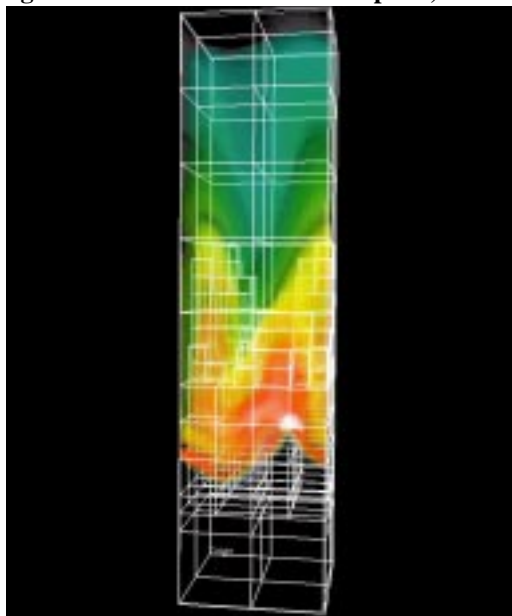
One troublesome dilemma is the speed difference between the infrastructure components and the problem size: disk transfer and network bandwidth rates are typically on the order of tens of megabytes per second, but data sizes are on the order of hundreds of gigabytes. How does one achieve interactivity on the desktop without moving all the data to the desktop?

Considering the visualization and rendering pipeline from Figure 1, we observe that in order to deploy a visualization tool on the desktop which is capable of rendering large data sets at interactive rates, the “object database” used by the renderer must be small enough to fit on the display platform. To that end, we have implemented a relatively new technique for volume rendering with a unique architecture that produces a relatively small object database, or scene graph<sup>4</sup>. As will be discussed later in the paper, we use a unique combination of task partitioning and parallelism to perform interactive volume visualization of large scientific data sets. Since visualization and rendering are pipelined and occur asynchronously, the viewer, which is “downstream” from the parallel software volume renderer, can interact with the render-

able objects at interactive rates. Updates of the scene graph through the visualization pipeline asynchronously from rendering, and occur at whatever rate the underlying infrastructure can provide.

A scene graph interface provides not only the means for parallel and asynchronous updates, but also an “umbrella” framework for rendering divergent data types. The scene graph system used in our implementation [8] supports storage and rendering of surface-based primitives (triangles, triangle strips, quads, polygons, etc.), vector-based primitives (lines, line strips), image-based data (volumes, textures, sprites and bitmaps), and text. The flexibility of this underlying infrastructure layer allows us to perform simultaneous rendering of volume and geometric data. Figure 3 is an image containing both volume rendering of density data, along with vector geometry (line segments) representing the adaptive grid created and used by the combustion simulation.

**FIGURE 3. Visapult Rendering of Combustion Data and Adaptive, Hierarchical Grids**



### 3.2 Parallel Volume Rendering Algorithm Taxonomy

Since volume rendering [9] is a computationally expensive and time consuming operation even with modest amounts of data, it is a likely candidate for parallelization. Algorithms for parallel volume rendering can be classified into two broad categories, *image order* and *object order*, based upon how the volume rendering task is decomposed across the pool of processors [10]. In an object order algorithm, the volume data is distributed across the processors using one of a number of different domain decomposition strategies (Figure 4). Each processor then renders its subset of the volume, producing an image. After all processors have finished rendering, the images from each processor must be gathered, then recombined into a final image. Recombination consists of image compositing using alpha blending [11], and must occur in a prescribed order (back-to-front or front-to-back). Note that each processor in an object order algorithm produces an intermediate image that may overlap in screen space with the images produced by other processors.

---

4. The term *scene graph* refers to a set of specialized data structures and associated services that provide management of displayable data and rendering services.

**FIGURE 4. Slab, Shaft and Block Decomposition**

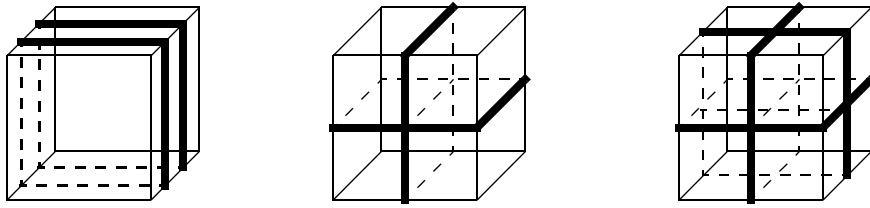


Image order algorithms, on the other hand, assign some region of screen space to each processor. The resulting images produced by each processor do not overlap, so recombination is not subject to an ordered image composition step. Depending upon the view, image order algorithms require some amount of data duplication across the processors, so do not scale as well with data size as the object order algorithms. The performance of image order parallel volume rendering algorithms is more sensitive to view orientation than the object order counterparts. In some views, there may be some processors with little or no work. In addition, as the model moves, the source volume data required at a given processor will change, requiring data redistribution as a function of model and view orientation.

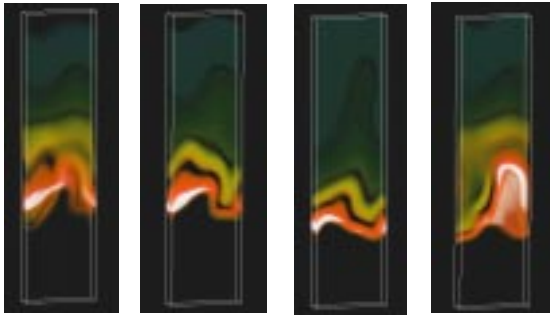
### 3.3 Image Based Rendering Assisted Volume Rendering

Image based rendering (IBR) methods [12, 13] have been the subject of much attention in recent years. IBR methods are used primarily for generating different views of an environment from a set of pre-acquired imagery. The properties of IBR which make it attractive include interactive viewing with low computational cost irrespective of scene complexity, and the ability to use images from either digitized photographs or rendered models. Common among IBR methods is a process of warping and blending images from known views to represent what would be seen from an arbitrary view.

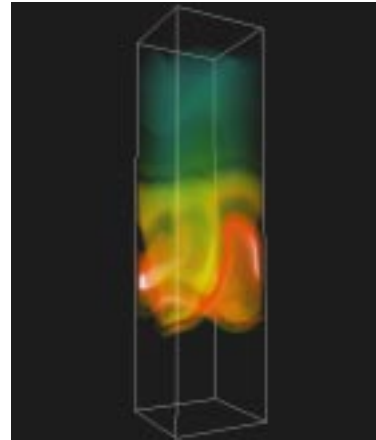
The concepts and principles of IBR model were recently applied to volume rendering [14]. Like the more conventional IBR counterparts, IBR assisted volume rendering (IBRAVR), seeks to achieve interactive rendering by avoiding the time-consuming process of completely rerendering the volume data for each frame. Instead, renderings of a model at arbitrary orientations are “computed” from “nearby” prerendered images. The prerendered images for the IBRAVR algorithm are obtained by volume rendering subsets of the entire volume. Using a slab decomposition, each source image would be obtained by volume rendering the slab of data. The total number of source images is equal to the number of data slabs created by data partitioning.

The per-frame, incremental rendering, or IBR component of IBRAVR, is implemented by using the pre-computed imagery as two dimensional textures which are texture-mapped onto geometry derived from the geometry of the slab decomposition, then rendered in depth order. In the basic algorithm, a single quadrilateral representing the center of the slab is used as the base geometry, and the computed imagery is texture mapped using alpha blending upon that geometry. With multiple slabs, there are multiple, overlapping, base geometries that are textured by the graphics hardware with the semi-transparent textures. As the model is rotated, the multiple textures correspondingly rotate in three dimensions, producing the impression of interactive volume rendering. As nearly all graphics hardware supports two-dimensional texturing, the IBRAVR viewer can be deployed on a wide variety of graphics platforms. An extension to this algorithm, described in [14], is replace the single quadrilateral with a quadrilateral mesh using offsets from the base plane for each point in the quad mesh. This enhancement will add a depth component to each of the IBR images, thereby enhancing the visualization process. We have included this extension in the Visapult implementation, but the details are omitted in this paper.

**FIGURE 5. IBR Assisted Volume Rendering**



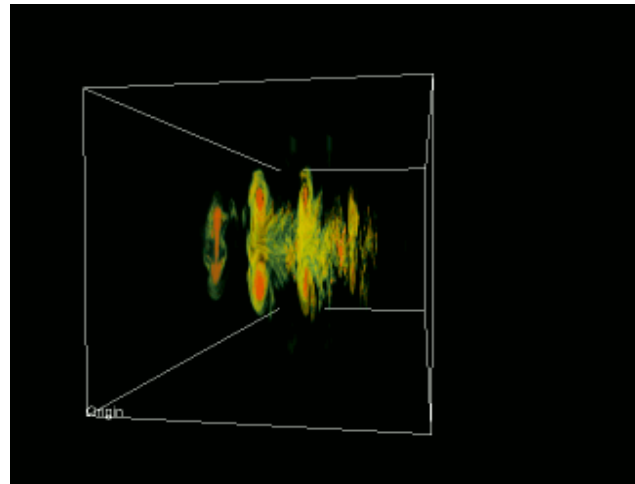
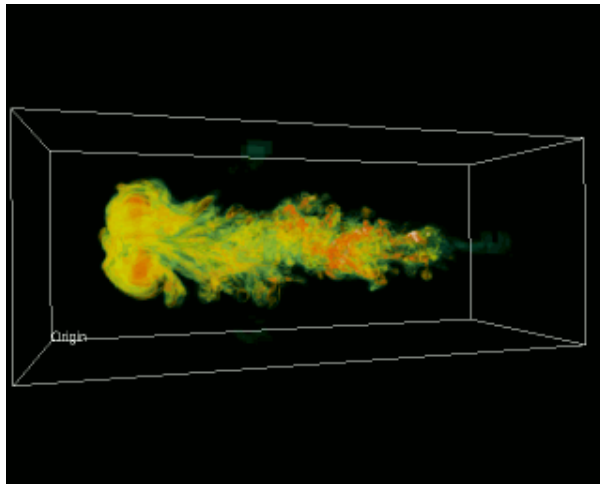
The source volume is subdivided into some number of slabs, each of which is volume rendered. The resulting images, along with geometric information derived from the original volume, are used as the source data for an IBR rendering engine.



The final IBR model can be interactively transformed without the need to perform costly volume rendering on each frame.

As described in [14], the IBRAVR model exhibits visual artifacts as the model is rotated away from an axis-aligned view (Figure 6). These artifacts result from volume subdivision along an axis-aligned view, but rendered using a view or orientation that is not “closely” axis aligned. As the model rotates away from an axis-aligned view, the artifacts become more pronounced. [14] reports that objects viewed within a cone of about sixteen degrees will appear to be relatively free of visual artifacts.

**FIGURE 6. IBRAVR Artifacts**



Using a nearly axis-aligned view, the IBRAVR method produces a high-fidelity image (left). When the model is rotated off-axis, visual artifacts can be seen (right). For the right image, we disabled axis-switching within Visapult, otherwise we would be viewing slices along the X-axis of the data.

Our implementation does not provide any remedies to this fundamental artifact of IBRAVR, but extends the base algorithm in a different manner that is useful for the purposes of visualization. On a per-frame basis, the Visapult viewer computes the best view axis, and transmits this information to the back end. The back end uses this information in order to select from either X-, Y-, or Z-axis aligned data slabs for use in volume rendering.

### 3.4 Visapult: Parallel and Remote IBRAVR

Visapult is a parallel and distributed implementation of an IBR assisted volume rendering engine. Our implementation can be thought of as a blend of an object-order parallel volume rendering engine with an IBRAVR viewer that uses a parallel, network-based data gathering model as an image assembly framework. The fundamental IBRAVR algorithm decomposes nicely into a distributed, pipelined and parallel architecture: a parallel object-order, parallel I/O capable volume rendering engine that produces images, and a parallel viewer that uses IBR techniques to assemble the individual images into a final display.

The Visapult back end reads raw scientific data from one of a number of different data sources, and each back end process performs volume rendering on some subset of the data, regardless of the viewpoint. The resulting images are transmitted to the Visapult viewer for final assembly into a model (scene graph), then rendered to the user. Owing to the IBRAVR design, the raw scientific data is distributed, or partitioned, amongst the back end processors using a slab-based decomposition (Figure 4). During the partitioning process, data is read into each processor in parallel. Each processor then performs software volume rendering upon its subset of the data. The resulting image from each processor is transmitted over the network to a peer receiver in the Visapult viewer, where it is inserted into the scene graph as a 2D texture.

On the viewer side, graphics interactivity results from a combination of the IBRAVR viewer model with a decoupling of scene graph updates from rendering. The amount of viewer-side data to be rendered is much smaller than the size of the raw volume data<sup>5</sup>, so even software-only graphics systems are not overwhelmed. To implement the decoupling of rendering from scene graph updates, the viewer itself is a multi-threaded application, with one thread dedicated to interactive rendering, and other threads dedicated to receiving data from the Visapult back end visualization processes over multiple simultaneous network connections (implemented with a custom TCP-based protocol over striped sockets). Except for a small amount of scene graph access control with semaphores, I/O and rendering occur in an asynchronous fashion, so all pipes are full, making effective use of network and computational resources. Additional architectural details of the Visapult back end and viewer are presented in Appendix A.

### 3.5 Visapult's Use of the LBL DPSS as a Data Cache

In its role as data collector, the Visapult back end fetches raw scientific data for the purpose of visualization. One source of data is the DPSS, which is used as a storage cache for data sets that are too large to fit on the workstation. These data sets, generated on supercomputers or clusters of workstations, are typically on the order of 30 to 100 GB, and are often stored on archival systems such as HPSS [15], a high performance tertiary storage system. Clearly, it is impractical to transfer data sets of this magnitude to a local disk for processing. Also, archival systems such as the HPSS are not typically tuned for wide-area network access, and only provide full file, not block level, access to data. The DPSS addresses both of these issues; it is optimized for wide-area access to large files, and provides block level access, eliminating the need to transfer the entire file across the network. Therefore, we can migrate the files from HPSS to a nearby DPSS cache.

The DPSS provides several important and unique capabilities for data intensive distributed computing environments. It provides application-specific interfaces to an extremely large space of logical blocks. It offers the ability to build large, high-performance storage systems from inexpensive commodity components. It also offers the ability to increase performance by increasing the number of parallel disk servers.

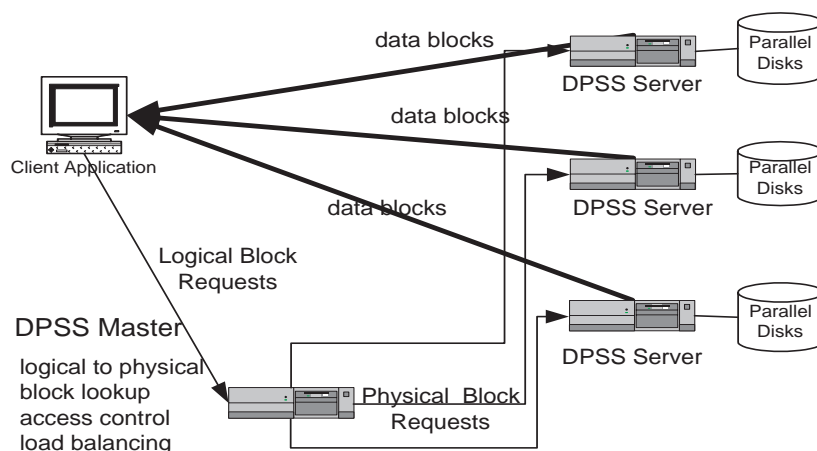
---

5. Where the size of the raw volume data is  $O(n^3)$ , the amount of data to be rendered in the viewer is  $O(n^2)$ .



Typical DPSS implementations consist of several low-cost workstations as DPSS block servers, each with several disk controllers, and several disks on each controller. A four-server DPSS with a capacity of one Terabyte (costing about \$15K in mid-2000) can thus deliver throughput of over 150 megabytes per second by providing parallel access to 15-20 disks. The overall architecture of the DPSS is illustrated in Figure 7.

**FIGURE 7. DPSS Architecture**



The application interface to the DPSS cache supports a variety of I/O semantics, including Unix-like I/O semantics, through an easy-to-use client API library (e.g., `dpssOpen()`, `dpssRead()`, `dpssWrite()`, `dpssL-Seek()`, `dpssClose()`). The DPSS client library is multi-threaded, where the number of client threads is equal to the number of DPSS servers. Therefore the speed of the client scales with the speed of the server, assuming the client host is powerful enough. This parallelism is leveraged by the parallel volume rendering performed by the Visapult back end.

### 3.6 Profiling and Performance Analysis - NetLogger

Profiling and analysis of an application's behavior and performance is an important part of the development process, but can prove challenging when the application consists of cooperative, distributed components. In our project, we made use of the NetLogger profiling toolkit for obtaining performance data from the application [16]. NetLogger includes tools for generating precision event logs that can be used to provide detailed end-to-end application and system level monitoring, and for visualizing log data to view the state of the distributed system. NetLogger has proven to be invaluable for diagnosing problems in networks and in distributed systems code. This approach is novel in that it combines network, host, and application-level monitoring, providing a complete view of the entire system.

The NetLogger system has a procedural interface: subroutine calls to generate NetLogger events are placed inside the source code of the application. Prior to running the application, a NetLogger daemon is launched on a host accessible to all components of the distributed application. During the course of application execution, the NetLogger subroutine calls communicate with the daemon host, where events are accumulated into an event log. This event log is then used as input for NetLogger visualization and analysis tools.

NLV, the NetLogger visualization tool, generates two dimensional plots from the raw data accumulated during a run. NetLogger and NLV were used extensively in Visapult field testing, and numerous examples of NLV output appear later in upcoming sections.

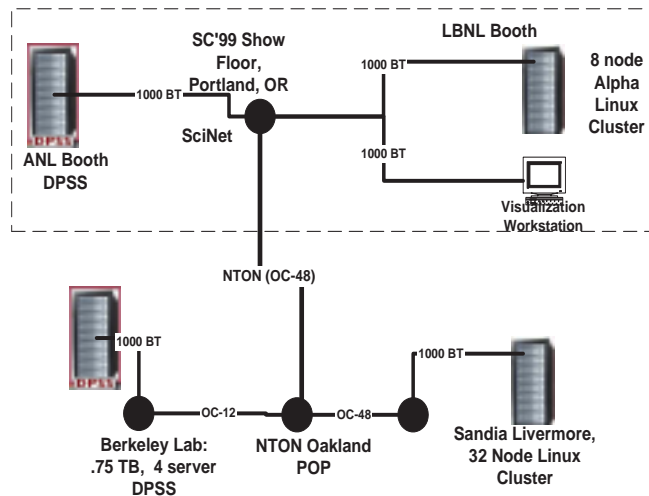
## 4.0 Visapult Field Testing and Evolution

In this section, we present several field testing experiments along with performance enhancements suggested by subsequent analysis. An early Visapult implementation was first presented at SC99 as part of a Research Exhibit. Since then, Visapult has become the reference application for a research program sponsored by the U.S. Department of Energy called *The Combustion Corridor*, and has been field-tested using several configurations of high speed testbed WANs using several different facilities. Research projects such as The Combustion Corridor seek to harness distributed resources for the purpose of scientific computing, such as high speed testbed networks, network storage systems, computational resources and large scale scientific data.

### 4.1 SC99 Research Exhibit

A preliminary version of Visapult was demonstrated at the SC99 conference in Portland, Oregon, reflecting a collaborative effort involving several research institutions: LBL, Sandia National Laboratory (SNL) and Argonne National Laboratory (ANL). Data from a cosmology hydrodynamic simulation<sup>6</sup> and a reactive chemistry combustion simulation<sup>7</sup> were transmitted over a WAN and visualized on the show floor at SC99. The demonstration required the use of NTON (National Transparent Optical Network) and SciNet99, the SC99 show floor network, to connect all of the resources (Figure 8).

FIGURE 8. Visapult SC99 Configuration



During the course of SC99, we used several different configurations of data sources, computational engines and networks as illustrated in Figure 8. Cosmology data was stored on DPSS systems at LBL and in the Argonne National Laboratory booth. Combustion data was stored on a parallel file system on the Cray T3E at the National Energy Research Scientific Computing Center (NERSC), located in Berkeley at LBL. Cosmology data was processed by a Visapult back end on the SNL CPlant [17] located in Livermore, California, or on the Babel Cluster [18] located in the LBL booth at SC99. The combustion data was pro-

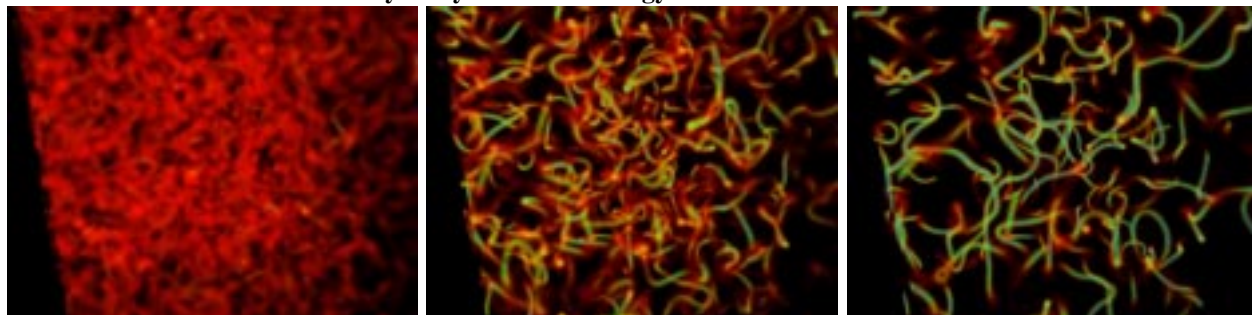
6. Cosmology data courtesy of Julian Borrill, Scientific Computing Group, National Energy Research Scientific Computing Center (NERSC), Lawrence Berkeley National Laboratory.

7. Combustion data courtesy of Vince Beckner and John Bell, Center for Computational Sciences and Engineering, National Energy Research Scientific Computing Center (NERSC), Lawrence Berkeley National Laboratory.

cessed by a Visapult back end running on the Cray T3E at NERSC in Berkeley. We also used multiple display devices for final rendering at SC99, including an ImmersaDesk located in the LBL booth, and a tiled surface display, located in the SNL booth. The ImmersaDesk allowed us to render the results in stereo. The tiled display system allowed us to demonstrate Visapult using a large-screen, theater-sized output format suitable for larger audiences.

We performed some preliminary analysis of the behavior of the system at SC99 using different network topologies and facilities. Our preliminary results showed that the majority of communication was between the DPSS (the network data cache) and the Visapult back end, with the link between the Visapult back end and viewer requiring much less bandwidth. This behavior is expected from the architecture of the system. Since the Visapult back end performs parallel volume visualization to reduce the data down to a small subset of images, it is expected that the amount of data resulting from the visualization and transmitted between the back end and viewer will be significantly less than the amount of data moved to the back end from the data source.

**FIGURE 9. Visualization of Hydrodynamic Cosmology Simulation Results at SC99**



We were capable of sustaining a data transfer rate of 250Mbps between the DPSS located at LBL and CPlant, and a rate of 150Mbps between the DPSS at LBL and the LBL cluster at SC99. The difference in transfer rates was based upon the different network topologies. The link between the SC99 show floor and LBL required resource sharing over SciNet.

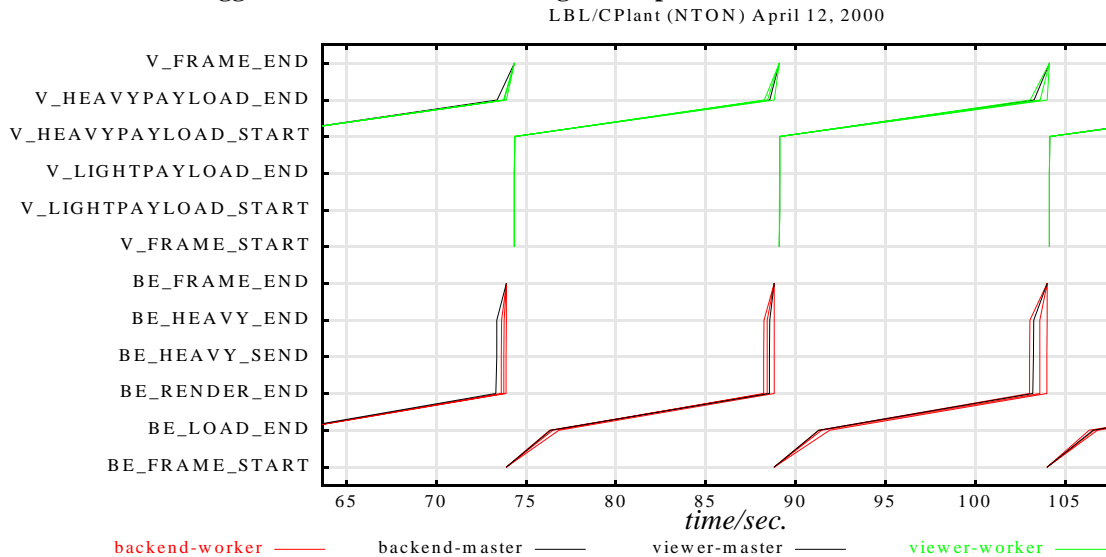
## 4.2 Combustion Corridor *First Light* Campaign

More recently, we have undertaken field testing using many of the same resources as for the SC99 project, but with an eye towards careful instrumentation and profiling analysis, and with larger data sets. This work is part of a project called *The Combustion Corridor*, sponsored by the U. S. Department of Energy, which is a collaborative research effort that includes LBL, ANL and SNL-CA. The term *Combustion Corridor* refers to the *process* of remote and collaborative visualization of large, scientific data sets for the Combustion Research community. The term “corridor” has been coined to refer to the metaphorical path from data source to human consumer, where the path spans geographical and system boundaries. A theme common across “corridor” projects is that many endeavors that were once possible only over LANs are now possible over WANs using a wider array of distributed resources. To a large extent, the needs and requirements of the Combustion Corridor are sufficiently general to be applicable to a wide variety of problem domains, including medicine, physics, and the geosciences.

Within the Combustion Corridor effort, we have performed several end-to-end runs using differing network topologies and platform configurations, which we refer to as “campaigns.” The first such campaign took place on 12 April 2000, and was a collaboration between LBL and SNL-CA. In this campaign, we used resources connected by NTON, a high speed testbed network. For this example, the raw scientific data was located on a DPSS at LBL in Berkeley, while the Visapult back end was located on the CPlant

Linux/Alpha cluster at SNL-CA. The Visapult viewer was running on a desktop machine at SNL-CA. The combustion simulation used for this example was from a 640x256x256 grid, and each grid value was represented with a single IEEE floating point number, for a total of 160 megabytes of data per time step for each of the 265 time steps. The theoretical limit of the network link is 622 Mbps, or the OC-12 connection between LBL and NTON.

**FIGURE 10. NetLogger Instrumentation/Profiling of Visapult**



For this image, profile data was collected from both the Visapult back end and viewer. The top row of traces, in green, represent the profile data from the viewer, while the bottom row of traces were obtained from the back end. The horizontal axis represents elapsed time from the start of the application. Each of the entries along the vertical axis of the code are strings associated with specific events, which occurred in order from bottom to top. The viewer events are prefixed with “V\_”, while the back end events have a “BE\_” prefix. Refer to Appendix A for additional details that will aid in interpretation of this data.

In Figure 10, we wish to draw attention to the performance profile of the Visapult back end performance shown by NetLogger instrumentation. The time required to load 160 megabytes of data into the back end from the DPSS over NTON was approximately three seconds<sup>8</sup>, for an approximate throughput rate of 433 megabits per second, which is in excess of the network performance realized during the SC99 demonstration over the same network link, reflecting improvements in the underlying Visapult implementation. The improvement in raw network performance was the result of a change to data staging and communications streamlining within Visapult. This amounts to a respectable 70% utilization rate of the theoretical bandwidth limit of the network while data was being transferred. The software rendering then consumed about eight or nine seconds on four processors of the CPlant cluster.

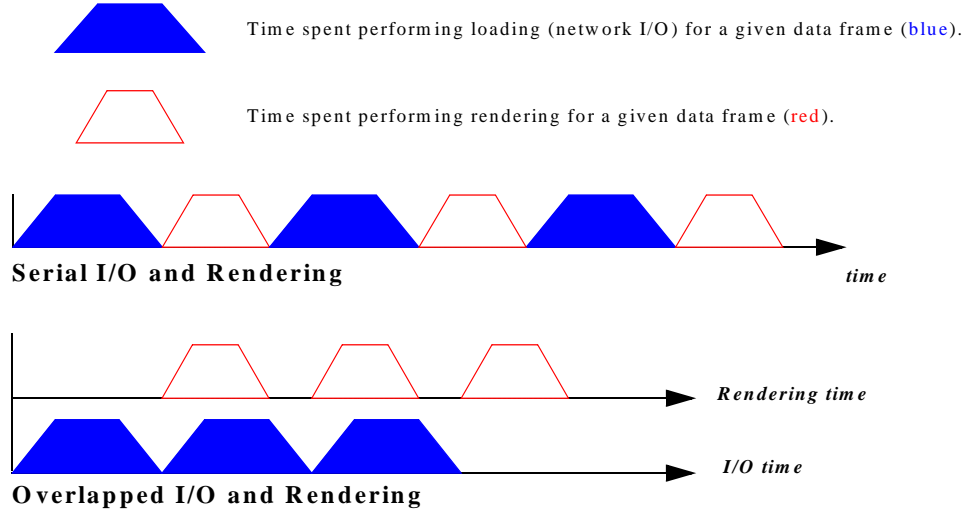
From this campaign, one significant design modification is suggested by the performance data - overlapping network transfers with rendering could have a significant positive impact upon the overall application performance. NetLogger performance profiles, such as that shown in Figure 10, are invaluable for identifying potential performance bottlenecks in distributed applications.

8. Displacement along the horizontal axis, time, between the tags BE\_FRAME\_START and BE\_LOAD\_END, which bracket the process of moving data from the DPSS into the Visapult back end on CPlant.

### 4.3 Overlapped I/O and Rendering

Each processing element (PE) in the Visapult back end loads a subset of a large scientific dataset, then volume renders it's subset of data. The resulting image is then transmitted to the viewer for use as a two-dimensional texture in a scene graph. Then, the process repeats, looping over time. If loading and rendering were overlapped, so as to occur simultaneously, then we would expect the overall application performance to significantly increase.

**FIGURE 11. Overlapped I/O and Rendering Timing Diagram**



In the discussion that follows, we refer to a *serial* implementation as one in which, in each PE of the parallel Visapult back end, rendering and data loading occur in a serial fashion. Note that even though we use the term *serial*, the back end is in fact a parallel job. Serial refers to how rendering and data loading are executed within each back end process. On the other hand, *overlapped* means that the process of rendering and data loading is implemented in a pipeline-parallel fashion, and occur simultaneously. Also, note that while the data for frame  $N$  is being rendered, data for frame  $N+1$  is being loaded.

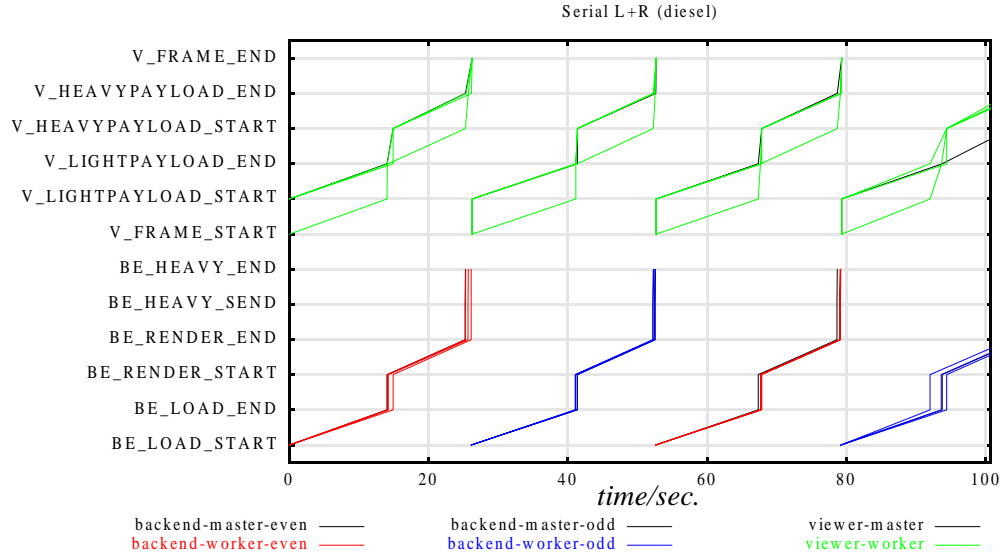
We can capture the behavior of both serial and overlapped versions, and estimate overall improvement as follows: let  $R$  be the time spent in each PE performing rendering for each of  $N$  timesteps of data (the red zones in Figure 11, above), and let  $L$  be the time spent by each PE loading data for each time step. The amount of time,  $T_s$ , required for  $N$  time steps' worth of data using the serial implementation is:  $T_s = N \cdot (L + R)$ . In contrast, the time required for  $N$  time steps using an overlapped implementation is:  $T_o = N \cdot \max(L, R) + \min(L, R)$ .

For illustrative purposes, if we assume that  $L$  and  $R$  are approximately equal, then the theoretical speedup realized using an overlapped implementation over one that is serial is  $T_s/T_o$ , or  $2N/(N+1)$ , which is nearly a 100 percent improvement. As the difference between  $L$  and  $R$  increases, the effective speedup resulting from an overlapped implementation will diminish. At one extreme, the overlapped implementation could be as much as nearly twice as fast as a serial implementation. At the other extreme, they will be nearly equal in performance.

The following two figures show the profiling results that compare serial and overlapped implementations of the Visapult back end data loading and rendering tasks. These tests were run using an eight processor Sun Microsystems E4500 server<sup>9</sup> connected to the LBL DPSS via gigabit ethernet (LAN), and were per-

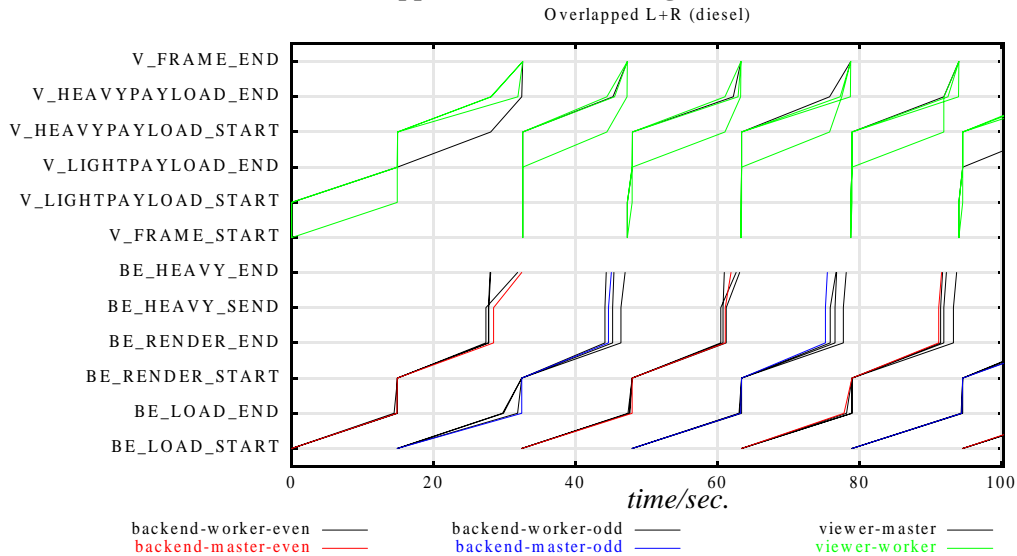
formed using ten timesteps from a large scientific data set. The serial implementation required approximately 265 seconds, while the overlapped version required approximately 169 seconds. In each case,  $L$  was approximately 15 seconds, while  $R$  was approximately 12 seconds.

**FIGURE 12. Execution Profile of Non-Overlapped I/O and Rendering**



Figures 12 and 13 were created using the NetLogger visualization tool, NLV, and graphically depict and contrast the performance of serial and overlapped implementations of the Visapult back end. In these Figures, the profile traces for the back end are colored according to data frame number; odd frames are blue while even frames are red.

**FIGURE 13. Execution Profile for Overlapped I/O and Rendering**



Note that in Figure 13, data loading for frame  $N+1$  and rendering for frame  $N$  commence simultaneously. In both the serial and overlapped tests, the time required for each of  $L$  and  $R$  are approximately equal. As

---

9. Eight, 336Mhz UltraSparcII processors.



we shall see in the next section, the time required for each of  $L$  and  $R$  in serial and overlapped implementations can vary as a function of the underlying architecture. Details of the overlapped implementation are presented in Appendix B.

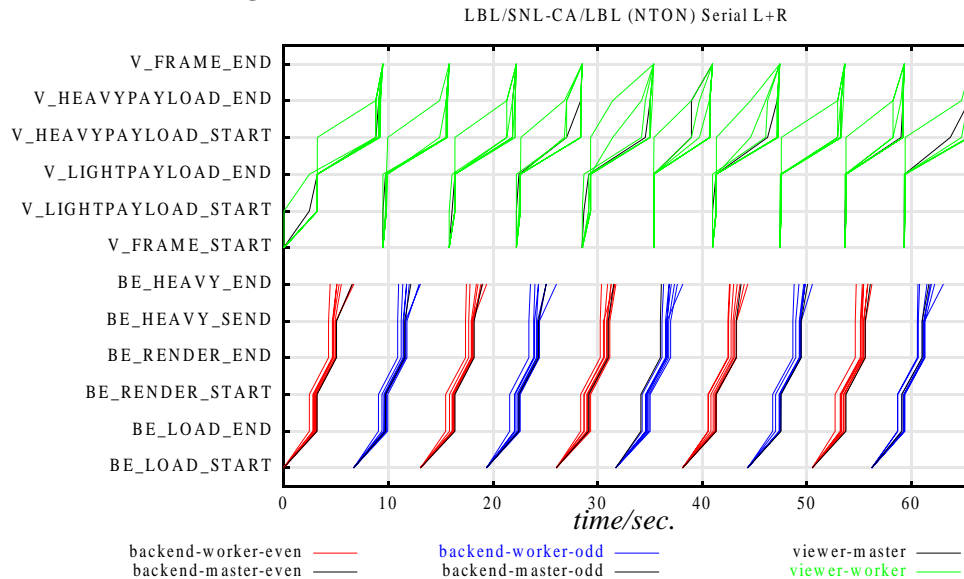
## 4.4 Further Combustion Corridor Testing

In this section, we present performance results obtained while executing Visapult over two different WANs and using two different compute platforms on the back end. One of the WANs, NTON, is a high-speed test-bed network that includes an OC12 path from LBL to SNL-CA. The other network, ESnet, is built atop an OC-12 backbone between LBL and ANL, but is a shared resource. The two compute platforms consist of a distributed memory Linux-Alpha cluster, and a large SMP. Each cluster node contains a pair of network interfaces: one for inter-node communication, and the other for external network access. The SMP uses a single gigabit ethernet interface for external network access, which is shared amongst all processors of the SMP. Our goals in the following tests are to obtain an estimate of network bandwidth utilization, and to compare the effect of serial and overlapped implementations of the Visapult back end on two different compute platforms.

### 4.4.1 LBL to CPlant over NTON

In the following two tests, we read data from a DPSS at LBL into CPlant nodes over NTON, performed parallel volume rendering on CPlant, then transmitted the resultant imagery to a viewer at LBL over ESnet. In the earlier campaign that used the LBL DPSS/CPlant/NTON combination (Figure 10), the back end did not yet support overlapped data loading and rendering. The profiles that follow compare and contrast the effect of serial and overlapped data loading and rendering. Figure 14 shows the performance profile of a serial implementation.

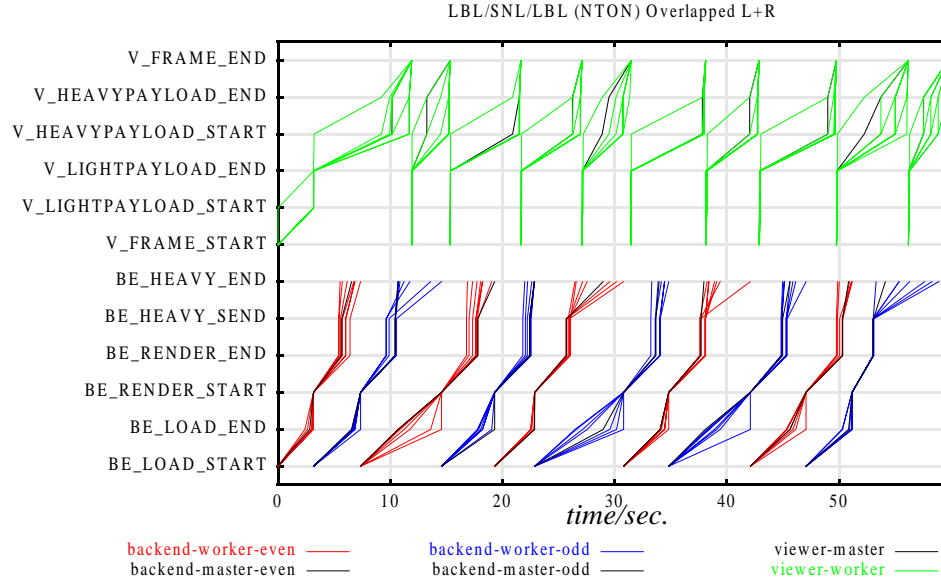
**FIGURE 14. Serial L+R on Eight CPlant Nodes**



In this example, we used eight nodes of CPlant, a Linux-Alpha cluster. Note that the time required to load 160 MB of data using eight nodes is approximately equal to the time required when using four nodes. From this, we observe that the use of additional nodes will not necessarily improve data throughput, as we have completely consumed all available network bandwidth. On the other hand, rendering time has been reduced to approximately half the time required when using four processors. Given the domain decomposi-

tion of the volume data, we expect linear speedup in the rendering process as the number of processors increases.

**FIGURE 15. Overlapped L+R on Eight CPlant Nodes**



The performance profile in Figure 15 was obtained by running a Visapult back end with overlapped data loading and rendering. One feature in Figure 15 that was expected, but difficult to characterize, is the increased time required for data loading, and the variability in load times from time step to time step. We can presume, based upon the results shown in Figure 15, there may be a relationship between the variability in completion times of transmission of image data from the back end to the viewer and the variability in data loading times<sup>10</sup>. The results indicate that as completion of transmission of outbound images becomes more staggered, inbound data loading is delayed. Another area of interest is CPU contention between the rendering and data loading processes. On CPlant, both rendering and data loading activities share a single CPU. While the render task is CPU intensive and the data loader is an I/O process, there appears to be a significant CPU demand incurred by the data loading process. This may be due in part to implementation details of the underlying network interface card (NIC) driver. It is widely known that some NIC drivers generate more interrupts than others, and these interrupts incur a cost in terms of CPU load. Some gigabit ethernet cards provide the option for using “jumbo frames” (9KB MTUs vs. 1.5KB MTUs), which incur lower interrupt overhead. However, using jumbo frames over a WAN is problematic.

#### 4.4.2 LBL to ANL over ESnet

The following two tests contrast serial and overlapped load and render operations on a large symmetric multiprocessing platform with shared memory (SMP)<sup>11</sup> located at ANL. The Visapult back end, running on the SMP at ANL read data from the DPSS at LBL over ESnet, then transmitted partial volume rendering results to a viewer located at LBL, also connected via ESnet. The ESnet link in these tests has a higher latency than the NTON link between LBL and SNL, and delivers an average bandwidth of approximately 100Mbps as measured with commonly available network tools, such as *iperf*<sup>12</sup>.

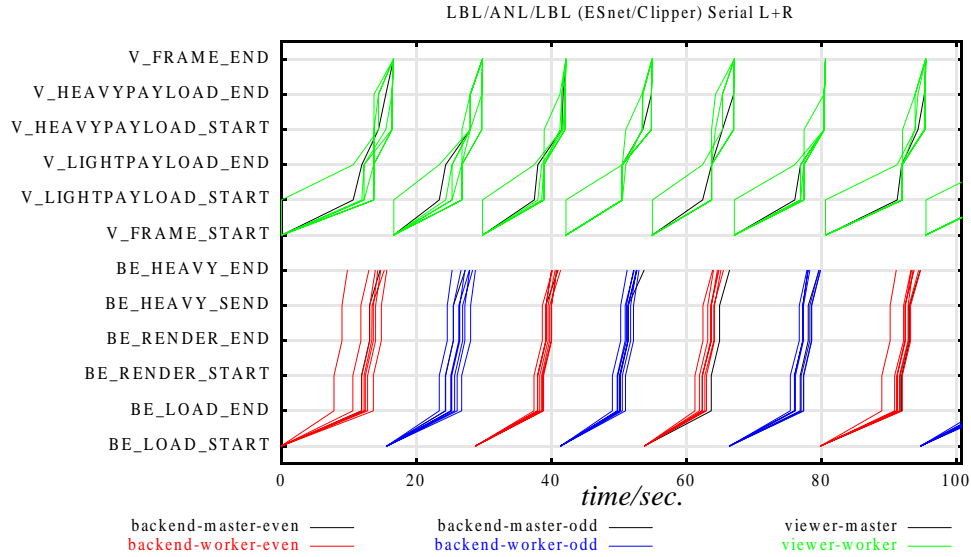
10. BE\_LOAD\_START and BE\_LOAD\_END bracket movement of data from the DPSS into each back end PE, while BE\_HEAVY\_SEND and BE\_HEAVY\_END bracket image transmission from the back end to the viewer.

11. A sixteen processor SGI Onyx2.



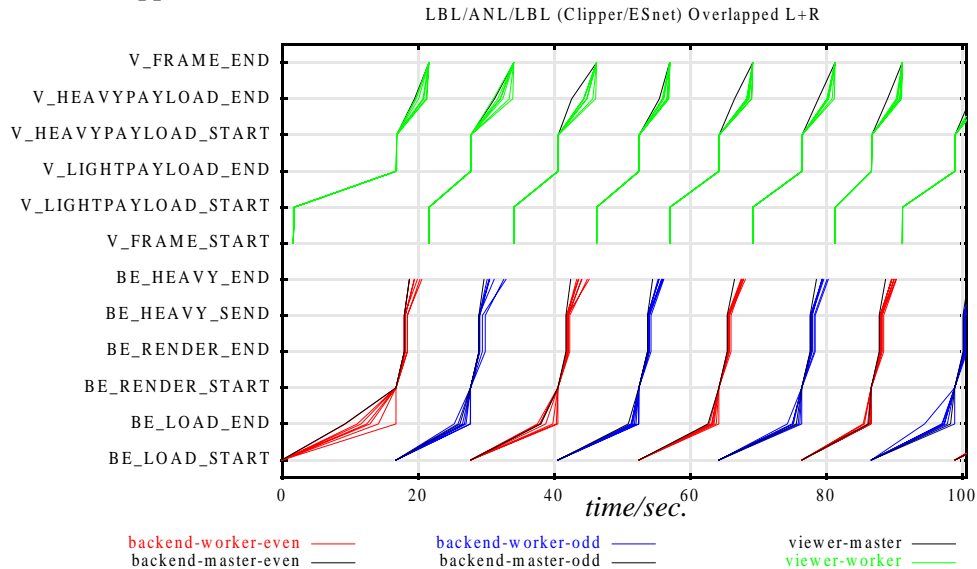
Figure 16 shows the performance profile of a serial Visapult back end running on eight processors of the SMP. We observe that approximately ten seconds is required to move 160 megabytes of data per data frame from the DPSS at LBL to ANL over ESnet, yielding a bandwidth consumption of about 128Mbps. Note that data loading time dominates in this case, owing to the significantly lower network capacity. We are able to achieve slightly better bandwidth utilization than a tool like *iperf* owing to the highly parallelized nature of our data loading.

**FIGURE 16. Serial L+R on an SMP**



Performance profile for the overlapped Visapult back end is shown in Figure 17. Similar to the NTON/CPlant tests, average elapsed time for overlapped data loading is slightly higher than the serial implementation. After the first time step's worth of data was loaded and the TCP window fully opened, we were able to steadily consume in excess of 100Mbps between the LBL DPSS and ANL over ESnet.

**FIGURE 17. Overlapped L+R on an SMP**



It appears that the SMP platform is better suited for the Visapult back end than a distributed memory platform: when each back end process, consisting of a rendering and a data loading thread, map directly onto a CPU, there appears to be less contention and context switching. In contrast, on the cluster, each of the two components of a single back end process must share a single CPU. In addition, the multiple NIC interfaces present on clusters present the possibility of achieving higher aggregate bandwidth utilization than the common SMP configuration of a single NIC shared by all nodes.

## 5.0 Future Work

We have obtained performance numbers for only a subset of contemporary architectures and available high speed testbed networks: large SMPs over a relatively slow and high latency network, and distributed memory systems with single CPU nodes as the compute platform over NTON, a high speed, low latency testbed network. Testing on additional compute platforms, particularly distributed memory architectures with multiple CPUs and shared memory on a single node, as well as an MPI-only implementation of the back end would serve to explore a significant portion of the platform-specific parameter space, and would serve to reveal additional strengths and weaknesses in the overall Visapult architecture.

Access to additional testbed networks is not a trivial task, and often requires the coordination of divergent research and operational groups. From the performance numbers shown in the previous section, it is clear that Visapult completely saturated all of the networks tested, and application throughput will be a function of the capacity of the underlying network. Despite completely using all available network bandwidth, the networks we tested do not have sufficient capacity to meet the challenges of terascale visualization. To put the problem into perspective, the time required to move our 265-timestep dataset (a total of 41.4 gigabytes) over NTON is on the order of eight minutes (a new timestep every 3 seconds), while over ESnet, the time required is on the order of 44 minutes (a new timestep every 10 seconds). A reasonable target rate would be, for this problem, five timesteps per second, requiring effective bandwidth on the order of fifteen times faster than our OC12 connection to NTON; approximately a dedicated OC192 link. This application points out the importance of having Quality of Service (QoS) (including bandwidth reservation) capabilities in future networks. In our testing we were able to completely saturate the WAN link in each network configuration. QoS is needed to insure that this application does not adversely affect other bandwidth-sensitive applications using the link, and to provide some minimum bandwidth guarantees to a Visapult session.

As a parallelized and pipelined implementation of IBRAVR capable of performing interactive volume visualization of large scientific data sets, Visapult's use of IBR-like rendering techniques corroborates the experiences of others who have sought to apply IBR to large model visualization. One such effort used IBR representations of complex geometry as the basis for distance-based model switching as a rendering acceleration aid for navigation through complex CAD models [19]. From a graphics perspective, an architecture built around an embedded scene graph core has proven to be successful in this project. As scene graph technology has been targeted at retained mode rendering of primarily geometric-based data, the question remains as to the applicability of this technology to general IBR techniques. More importantly, the Visapult implementation highlights the relevance of embedded rendering technology within the context of network-based 3D graphics and visualization. Although there are many examples of emerging commercial 3D web-based applications, these tend to use VRML [20] as a medium for data exchange. VRML is a data storage format with an emphasis upon surface and vector geometry. More recently in the VRML97 and Web3D efforts, the VRML base extends geometric modeling to include sound and asynchronous "sensors" that generate events to be consumed and processed by the VRML browser. VRML as a data format doesn't appear to readily lend itself for use by distributed IBR applications: IBR allows for navigation through environments where the source is either precomputed or acquired imagery. We envision interesting future 3D, web-based applications that use the notion of navigating through environments constructed from acquired, rather than computed imagery.

In our experience, remote resource access and management can be a troublesome and tedious endeavor. One of the appealing themes in Corridor projects is the ability of a user to transparently take advantage of remote and distributed resources, such as network storage caches and computational facilities, without specialized knowledge about the distributed resources: access to testbed networks may require modifications to routing tables; the ability to launch a parallel job likely requires shell access to the remote resource; and access to DPSS systems is typically provided on an as-needed basis. In order for research scientists to successfully use a tool like Visapult, they may need detailed technical knowledge of networks, knowledge of the existence of and access to the remote resources, and must be capable of diagnosing the inevitable difficulties that arise when attempting to launch multiple components of a distributed application. Users want tools that are easy to use and help them accomplish their work. A good deal of our future work will be focused upon simplifying the access to and use of the remote and distributed resources upon which Visapult is built.

In this project, the DPSS has proven to be a useful tool. Storage systems of this type present an economical and scalable storage solution that will assume an increasingly important role in a network-centric computing environment. We expect that by augmenting the block data services with additional processing capabilities, the DPSS will become even more useful. For example, “wire level” compression would benefit a wide array of applications. In the case of lossy compression techniques, the degree of lossiness could be a function of network line parameters and under application control. Additional possibilities include off-line visualization services, such as the offline and automatic creation of thumbnail representations of datasets or metadata.

## **6.0 Conclusion**

Remote and distributed visualization and rendering algorithms increasingly depend upon a foundation of data management and data movement. As a Corridor project, Visapult has demonstrated the feasibility of using combinations of distributed resources, such as parallel network data caches and computational resources. A unique combination of data staging, parallel rendering and parallel I/O has produced a prototype application and framework that is capable of performing interactive visualization of large scientific data sets. Several instrumented test cases have shown that Visapult is capable of saturating the fastest high speed testbed networks available today. Despite these results, we conclude that these networks are still inadequate for the purposes of tera-scale visualization. Access to the networks can be troublesome, and applications such as Visapult can benefit from related research projects, such as QoS and bandwidth reservation to streamline access to and use of these emerging resources.

## **7.0 Acknowledgement**

This work was supported by the Director, Office of Science, Office of Basic Energy Sciences, of the U.S. Department of Energy under Contract No. DE-AC03-76SF00098. Special thanks to Helen Chen, Jim Brandt, Pete Wyckoff, and Mike Hertzner at Sandia National Laboratories for providing access to CPlant and for providing extraordinary support for this project. The scientific data sets used in our experiments were generated by and used with the permission of Julian Borrill, Scientific Computing Group, NERSC and Vince Beckner and John Bell at the Center for Computational Science and Engineering, also at NERSC. Access to computing facilities at Argonne was provided by Rick Stevens and Mike Papka of the Math and Computing Sciences Division at Argonne National Laboratory.

## 8.0 References

- [1] “A Network-Aware Distributed Storage Cache for Data Intensive Environments”, Tierney, B. Lee, J., Crowley, B., Holding, M., Hylton, J., Drake, F., Proceedings of IEEE High Performance Distributed Computing conference, August 1999, LBNL-42896. see: <http://www.didc.lbl.gov/DPSS/>
- [2] Abilene: <http://www.internet2.edu/abilene/>
- [3] Supernet: <http://www.ngi-supernet.org/>
- [4] ESnet: <http://www.es.net/>
- [5] National Transparent Optical Network (NTON): <http://www.ntonc.org/>
- [6] The Message Passing Interface (MPI) Standard, <http://www.mcs.anl.gov/mpi/>
- [7] “Evaluating Stereo and Motion Cues for Visualizing Information Nets in Three Dimensions,” C. Ware and G. Franck, ACM Transactions on Graphics, 15, 2, April 1996, pp. 121-140.
- [8] OpenRM Scene Graph, <http://openrm.sourceforge.net/>
- [9] “Volume Rendering,” R. Drebin, L. Carpenter, P. Hanrahan, in Proceedings of Siggraph 1988, Computer Graphics, Volume 22, Number 4, pp. 65-74.
- [10] “Communication Costs for Parallel Volume Rendering Algorithms,” Ulrich Neumann, IEEE Computer Graphics and Applications, Volume 14, Number 4, pp 49-58, July 1994.
- [11] “Compositing Digital Images,” T. Porter and T. Duff, in Proceedings of Siggraph 1984, Computer Graphics 18, Volume 3, pp. 253-260.
- [12] “Modeling and Rendering Architecture from Photographs: A Hybrid Geometry- and Image-Based Approach,” P. Debevec, C. Taylor, and J. Malik, Proceedings of Siggraph 1996, In *Computer Graphics Proceedings*, Annual Conference Series, 1996, ACM SIGGRAPH, pp. 11-20.
- [13] “Light Field Rendering,” M. Levoy and P. Hanrahan, Proceedings of Siggraph 1996, In *Computer Graphics Proceedings*, Annual Conference Series, 1996, ACM SIGGRAPH, pp 31-40.
- [14] “IBR Assisted Volume Rendering”, K. Mueller, N. Shareef, K. Huang, R. Crawfis, in Proceedings of IEEE Visualization 1999, Late Breaking Hot Topics, October 1999, pp 5-8.
- [15] The High Performance Storage System (HPSS), <http://www.sdsc.edu/projects/HPSS/hpss1.html>
- [16] “The NetLogger Methodology for High Performance Distributed Systems Performance Analysis”, B. Tierney, W. Johnston, B. Crowley, G. Hoo, C. Brooks, D. Gunter, Proceedings of IEEE High Performance Distributed Computing Conference, July 1998, LBNL-42611. see: <http://www.didc.lbl.gov/NetLogger/>
- [17] CPlant (Computational Plant), <http://www.cplant.ca.sandia.gov/>.
- [18] “Production Linux Clusters: Architecture and System Software for Manageability and Multi-User Access”. W. Saphir, P. Bozeman, R. Evard and P. Beckman. SC ‘99 Tutorial on 11/14/99. Available at [http://www.nersc.gov/research/ftg/tribble/production\\_linux\\_clusters\\_v1.pdf](http://www.nersc.gov/research/ftg/tribble/production_linux_clusters_v1.pdf)

[19] “MMR: An Integrated Massive Model Rendering System Using Geometric and Image-Based Acceleration,” D. Aliaga, et. al., in Proceedings of 1999 ACM Symposium on Interactive 3D Graphics.

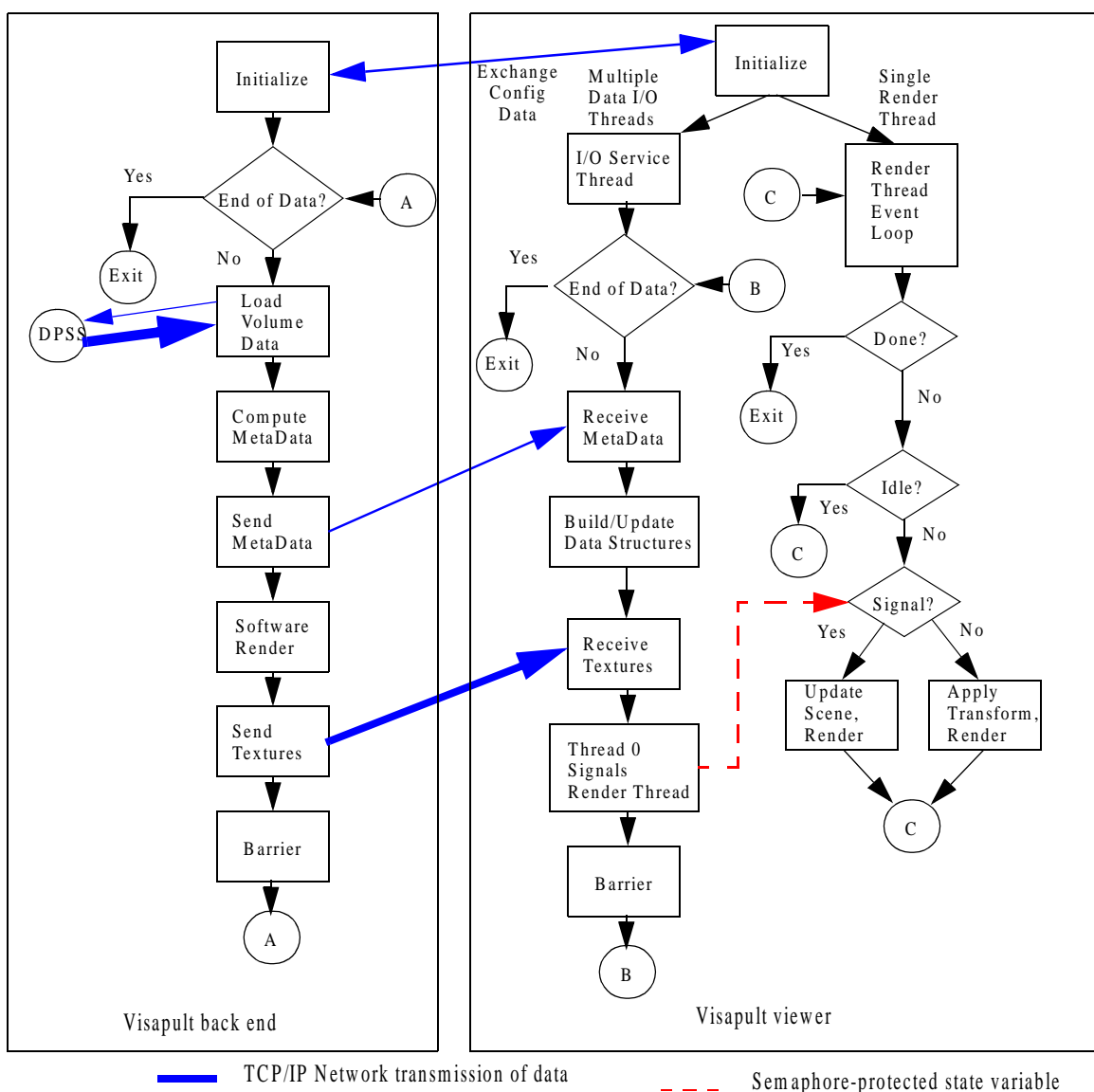
[20] <http://www.vrml.org/>

## 9.0 Appendix A - Visapult Internal Architecture

In this Appendix, we provide technical details about the internals of both the Visapult back end and viewer relevant to interpreting the plots of NetLogger profile data.

We begin with a flowchart-like depiction of the Visapult viewer and back end. The flowchart highlights coarse-grained tasks for both the viewer and back end, as well network communication between the cooperative processes.

**FIGURE 18. Visapult Architecture**



The following two tables provide additional detail about each of the tags present in the profile graphs generated by NetLogger. These tags are used in Figures 10, 12, 13, 14 and 15.

**TABLE 1. Visapult Viewer NetLogger Tags in NLV Figures**

Tag	Remarks
V_FRAME_START	Top of loop inside each thread that services an I/O connection with the back end. In the current implementation, the number of time steps, or loops, is set before each of these threads is launched at initialization time.
V_LIGHTPAYLOAD_START	Beginning of receipt of visualization metadata. Visualization metadata consists of texture size, bytes per pixel, and geometric information used to place the texture in a 3D scene. Visualization metadata is on the order of 256 bytes, hence the name “light payload.”
V_LIGHTPAYLOAD_END	Visualization metadata received.
V_HEAVYPAYLOAD_START	Beginning of receipt of visualization data. This data consists of raw pixel data, as well as any geometric data, such as triangles, boxes, and so forth. In our tests thus far, the size of this data is also relatively small compared to the size of the source volume. In this implementation, each thread receives a single texture, and while the size of the texture is a function of the resolution of the source volume, a typical size is on the order of 0.25 to 1.0 megabytes per texture. Geometric data is typically tens of kilobytes for the AMR grid data per timestep.
V_HEAVYPAYLOAD_END	All visualization data received.
V_FRAME_END	End of processing of this time step’s worth of data.

**TABLE 2. Visapult Back End NetLogger Tags in NLV Figures<sup>a</sup>**

Tag	Remarks
BE_LOAD_START	Each back end PE is about to load it’s subset of volume data.
BE_LOAD_END	Volume data load and format conversion completed. In our examples, this step includes loading of AMR species and grid data.
BE_LIGHT_SEND	Start transmitting visualization metadata to the viewer.
BE_LIGHT_END	Metadata transmission complete.
BE_RENDER_START	Start of parallel volume rendering process.
BE_RENDER_END	All rendering complete.
BE_HEAVY_SEND	Start transmitting visualization data. In this implementation, the visualization data consists of a single texture per back end PE, and optional geometric data representing the grid, and an optional elevation/offset map which the viewer will use to create a quadmesh.
BE_HEAVY_END	End of visualization data transmission.

a. There are many more NetLogger tags present in the Visapult back end. Many were omitted from this table, and from the figures, for brevity. These additional tags are useful for more detailed analysis of execution profiles within each large-grained task (e.g., “load data”).

## 10.0 Appendix B - Overlapped Visapult Back End Implementation Details

The Visapult back end is implemented using MPI as the multiprocessing and IPC framework. Each PE of the back end is responsible for reading a subset of the volume data, for rendering its subset of data, and for transmitting the rendering results to the Visapult viewer.

To implement overlapped data loading and rendering in each back end PE, the base MPI code was extended to launch a detached execution thread. We chose to use *pthread*s as the threading API due to its portability and wide availability. In the discussion that follows, we refer to the combination of a single MPI process and its associated detached, reader thread as a *process group* for the sake of clarity. The *reader thread* is the detached, freely-running pthread, and the *render process* is the MPI process. A flowchart of these cooperative processes is shown in Figure 19.

Upon entry, each MPI PE launches a detached, freely-running execution thread (reader thread). This thread logically executes concurrently with the MPI process. *Concurrent logical execution* means that we yield scheduling control to the host system. On distributed memory systems, such as Linux clusters, both reader thread and render process share a single CPU, thereby inviting contention. On SMP systems with a sufficient number of CPUs, in our experience, CPU contention appears to be minimized, if not eliminated.

In our implementation, the reader thread is a worker, and controlled by the render process. Each back end render process creates a pair of SystemV shared memory semaphores prior to launching the reader thread. Each of the semaphore pairs is shared by each render/reader process group, with one such pair for all MPI PEs. One of the semaphores, which we'll call *semaphore A*, is considered as an execution barrier from the perspective of the reader thread, while the other, *semaphore B*, is considered as an execution barrier from the perspective of the render process.

Upon entry to the reader thread, after some internal initialization occurs, the reader thread blocks waiting to gain access to *semaphore A*. The render process will request that either data from a specific time step will be read, or will request reader thread termination due to completion of all time steps. Once the reader thread gains access to *semaphore A*, it will examine the control variable (in shared memory) and take the appropriate course of action, either reading more data or exiting. Upon completion of the requested activity, the reader thread will post to *semaphore B*, then block awaiting access to *semaphore A*.

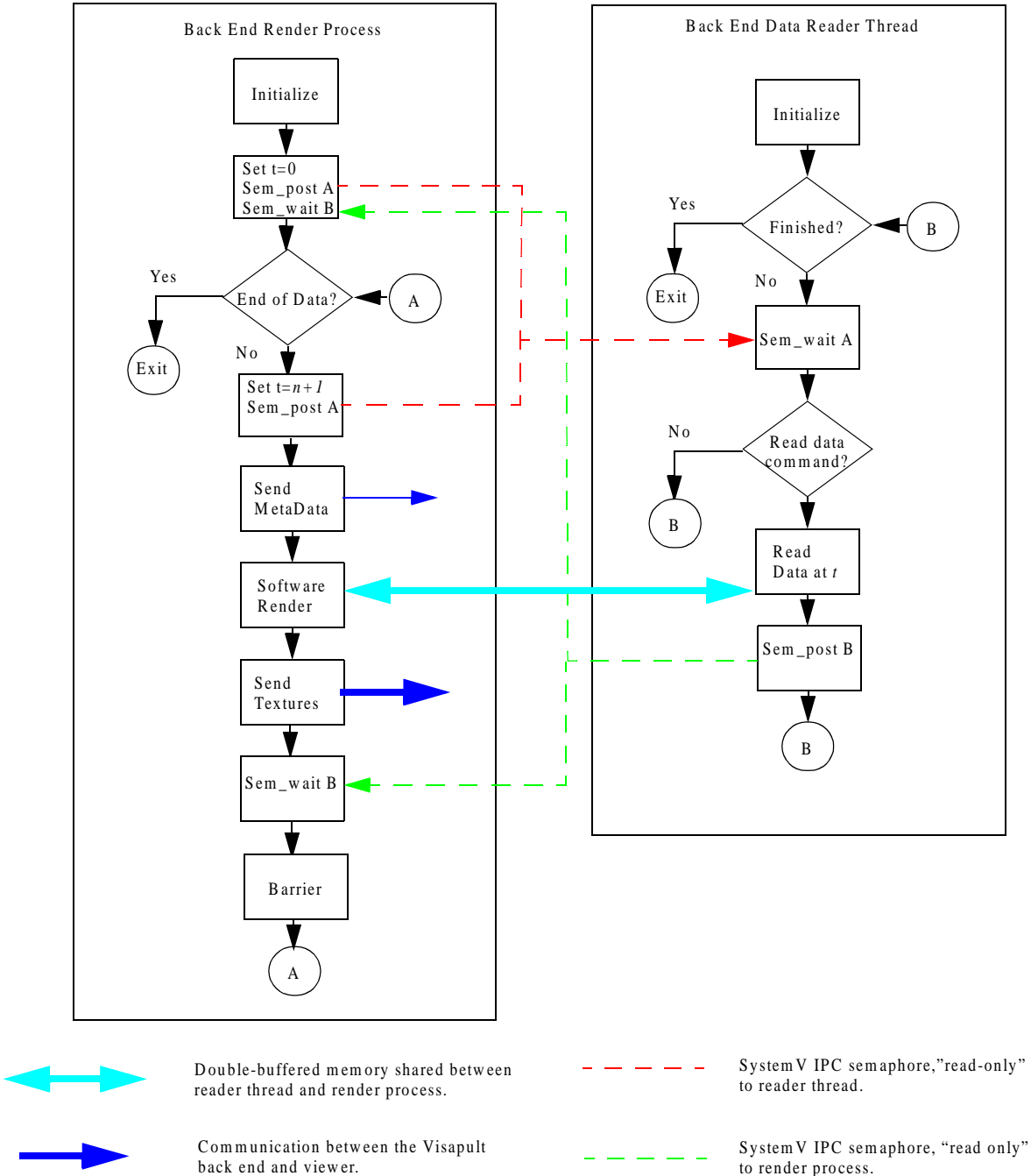
On the render process side, data from time step zero is first requested from the reader thread. Once that data has been loaded and is available, data from time step one is requested, and the render process begins to render data from time step zero. Once rendering is complete, results are transmitted to the viewer, then the render process will block while attempting to gain access to *semaphore B*. Upon gaining access to *semaphore B*, the render process requests the next time step's worth of data, and posts to *semaphore A*. This process continues until all the time varying data has been read, rendered and results transmitted to the viewer.

In addition to the control semaphores, a large block of memory is shared between reader thread and render process. The reader thread will load the raw scientific data into this large memory block during reading. This memory is considered to be *double-buffered*: its size is twice that of a single time step's worth of data, and the reader thread will use one half of the buffer for writing into, while the render process reads from the other half. Access control is implicit as a function of the time step using an even-odd decomposition. Due to the control architecture of the reader thread and render process, we are guaranteed that reader and render threads will not access the same odd/even data buffer at the same time.

We chose to extend the MPI base using pthreads in order to take advantage of the shared-memory model employed by threaded code. An alternative would be to use MPI-only constructs. For example, even-num-

bered processes would render, while odd-numbered processes would read data. The synchronization between the two would be similar, but using MPI constructs rather than SystemV semaphores. Of greater concern would be the need to transmit large amounts of scientific data between reader and render processes. We consciously chose to avoid incurring this additional cost by using a threaded model. In doing so, we may have incurred a penalty in the form of increased contention on distributed memory architectures with single-CPU nodes.

**FIGURE 19. Architecture of Overlapped Visapult Back End**

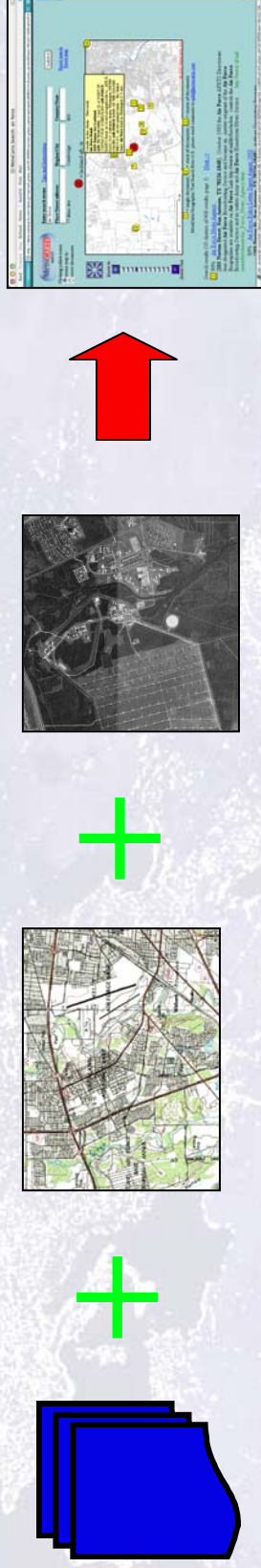




# MetaCarta

## Geographic Text Search

All-Source Intelligence + MetaCarta = Success.



MetaCarta pinpoints text messages geographically.

Initially funded by DARPA, MetaCarta allows decision makers and analysts to quickly visualize and manipulate intelligence.



Enter search terms [Tips and Instructions](#)

zulu:0800-0930 date:170302 air force

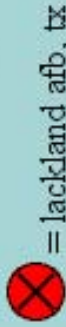
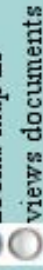
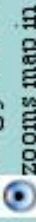
Submit

Place/Street address Region/City Country/State

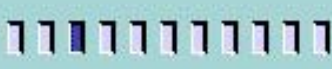
77 Mass Ave Cambridge MA

[Reset search](#)  
[Reset map](#)

Clicking yellow icons:



Zoom In



Zoom Out



1 = single document, 1 = stack of documents; 1 = cluster of documents

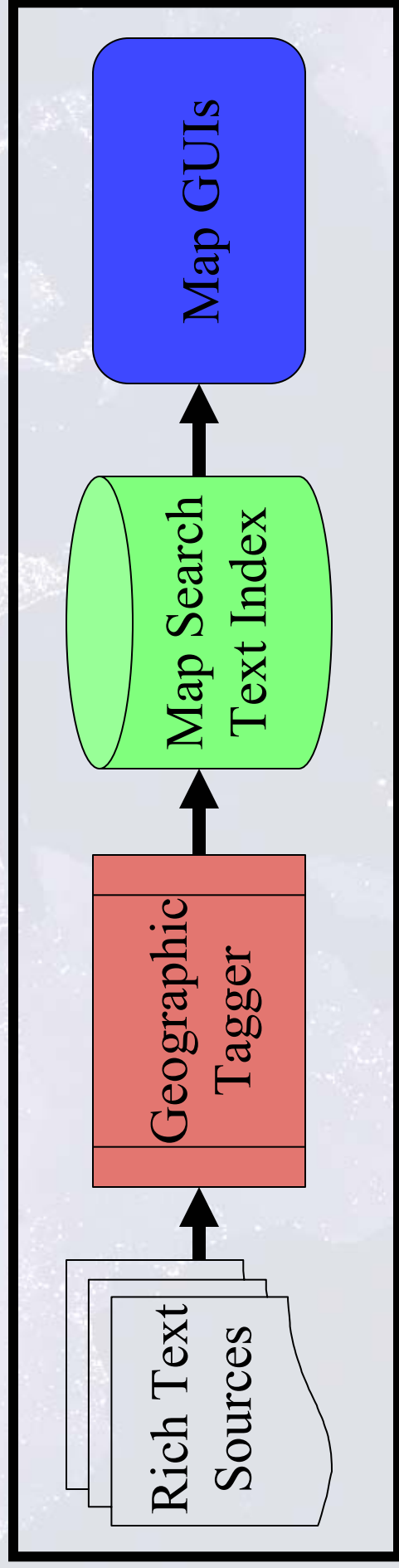
MetaCarta Geographic Text Search Beta v1.0 - please send comments to [info@metacarta.com](mailto:info@metacarta.com)



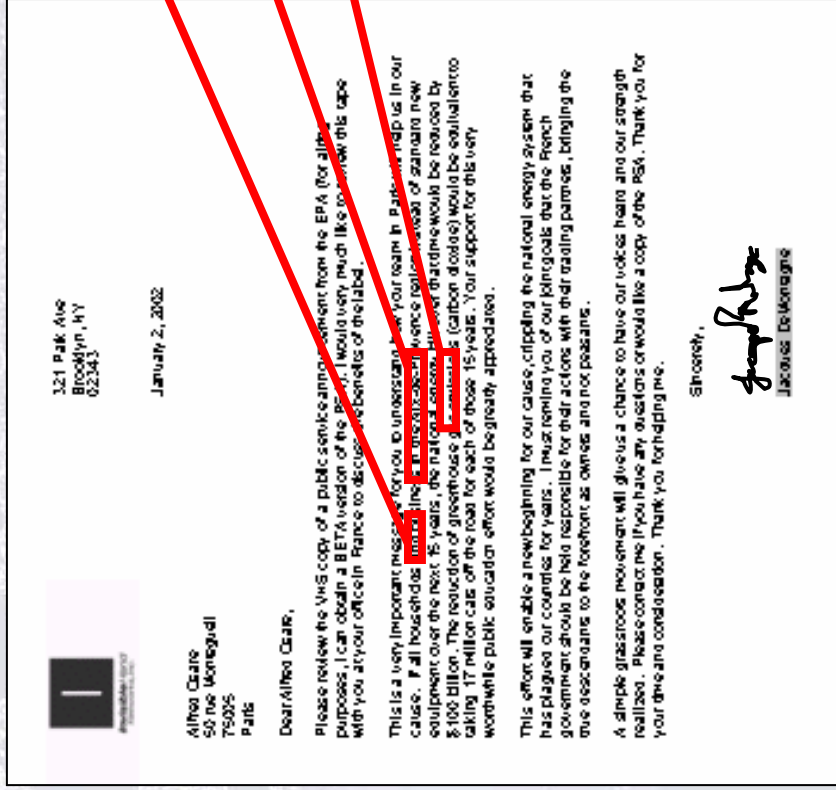
# Block Diagram

- Automatically tag geographic references.
- Index documents and databases for fast mapsearch.
- Explore previously unmapable data with a map.

95



# Auto Tagging of Geo References



France...

Aix-en-Provence...

St. Sauveur...

(43.534541°, 5.438329°, 88%)

The MetaCarta *Geographic Tagger* automatically extracts geographic references from rich text to create latitude, longitude, and confidence based on linguistic context.

# What's so hard about geo-tagging?

- “Media” is a place in Morocco, Bangladesh, Liberia, Uganda, Philippines.
- Which “Paris” did you mean? Paris, Texas?

Standard gazetteers do not have all the names we need, so we use machine learning to extract new names from massive open sources. For example:

- **“Karte Bokhti”** = (74.75349°, 43.75993°, 95%), neighborhood Mazar-I-Sharif
- **“Volpe Center”** = (42.369970°, -71.105249°, 100%), DOT facility Cambridge, MA
- **“jrf@mit.edu”** = (42.393161°, -71.115577°, 80%), home address used in newsgroups

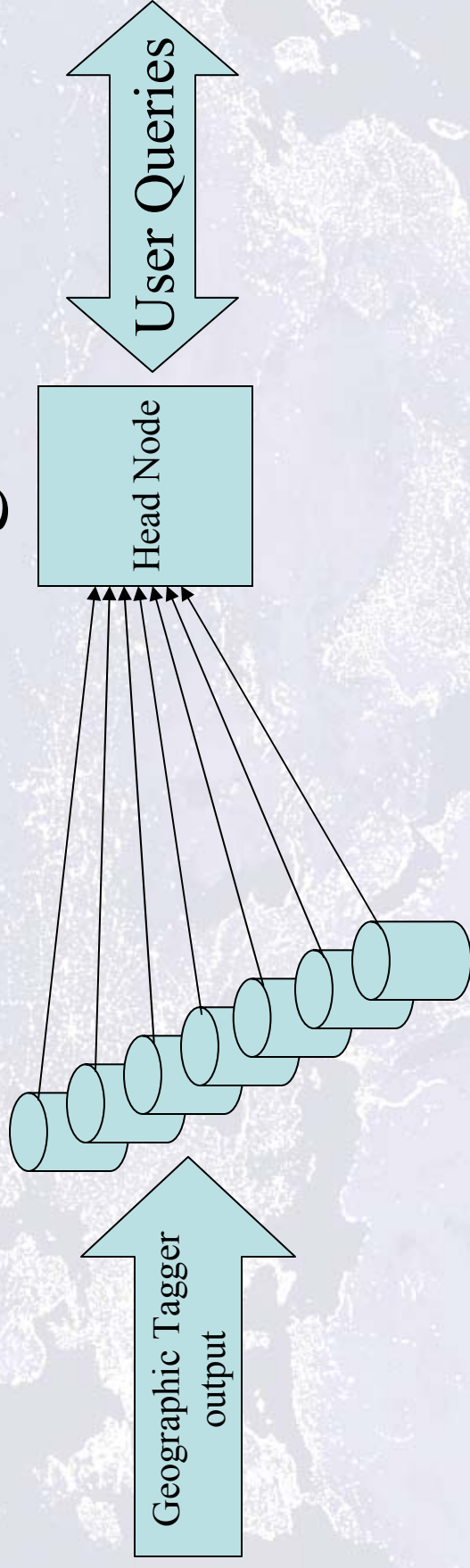




# What can you do with geo-tagged messages?

# Index them for search!

# Full-Text Indexing for Search



99

Output from the *Geographic Tagger* can be stored in the patent-pending *Map Search Text Index*. These revolutionary algorithms combine maps and full-text information without slow database joins.

The *Map Search Text Index* is parallelized into database slices to support unlimited volumes of data. Slices can run on a physically separated machine.



# Plug-n-Play Browser Search GUI

100

Flexibly combine reports, maps, and overhead.



Clicking yellow icons:  
• zooms map in  
• views documents

Enter search terms [Tips and Instructions](#)

Enter keywords

Submit

Place/Street address

Region/City

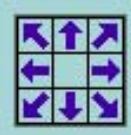
Country/State

77 Mass Ave

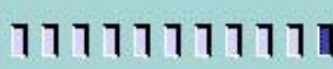
Cambridge

MA

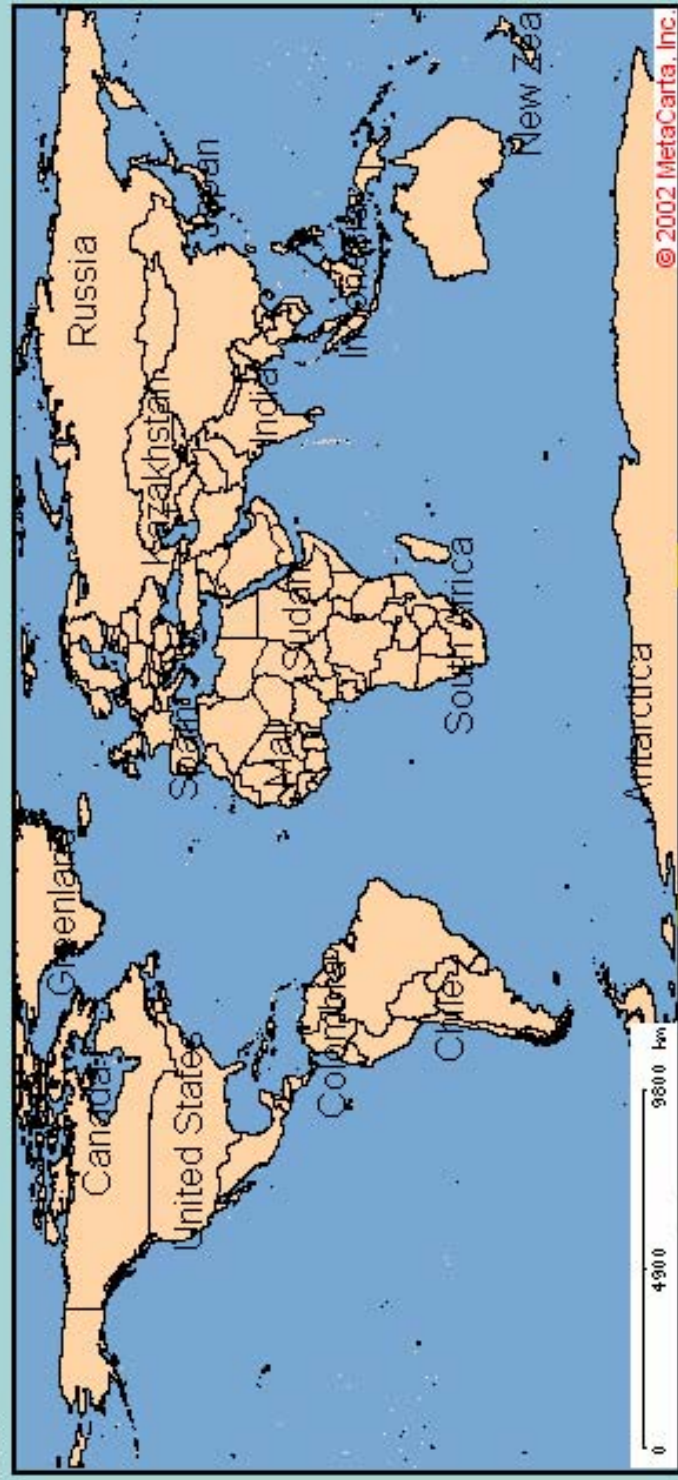
[Reset search](#)  
[Reset map](#)



Zoom In



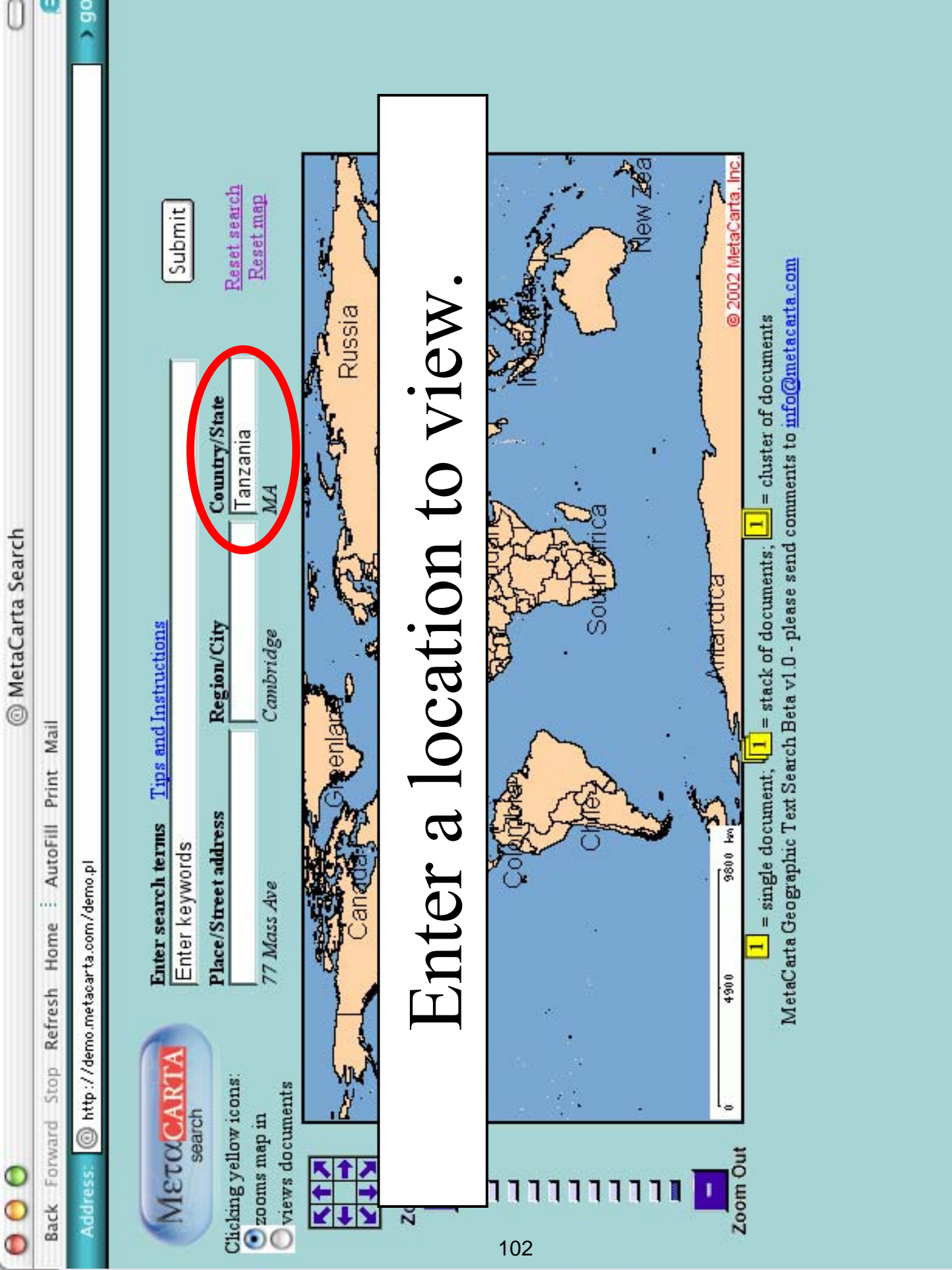
Zoom Out



© 2002 MetaCarta, Inc.

**1** = single document, **1** = stack of documents, **1** = cluster of documents

MetaCarta Geographic Text Search Beta v1.0 - please send comments to [info@metacarta.com](mailto:info@metacarta.com)



Clicking yellow icons:

zooms map in  
views documents



Enter a location to view.

Zoom Out

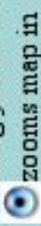
1 = single document; 1 = stack of documents; 1 = cluster of documents

MetaCarta Geographic Text Search Beta v1.0 - please send comments to [info@metacarta.com](mailto:info@metacarta.com)





Clicking yellow icons:



zooms map in



views documents

Enter search terms [Tips and Instructions](#)

Enter keywords

Submit

Place/Street address

Region/City

Country/State

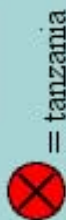
77 Mass Ave

Cambridge

MA

[Reset search](#)

[Reset map](#)



Zoom In



Zoom Out



1

1

1

= single document; = stack of documents; = cluster of documents

MetaCarta Geographic Text Search Beta v1.0 - please send comments to [info@metacarta.com](mailto:info@metacarta.com)

BackForwardStopRefreshHomeAutofillPrintMail

© MetaCarta Search

Address: http://demo.metacarta.com/demo.pl?current\_place\_lat=-5.802823&current\_place\_lon=tanzania&zoom\_level=7&lon=35.440099&lat=-5.802823

MetaCarta search

Clicking yellow icons:  
zooms map in  
views documents

Enter search terms  
volcan

Tips and Instructions

Submit

Reset search  
Reset map

Place/Street Address  
77 Mass Ave

Region/City  
Cambridge

Country/State  
MA

Zoom In

Zoom Out

Map navigation icons

Map of East Africa

© 2002 MetaCarta, Inc.

Enter search terms to find documents.

1 = single document,  
1 = stack of documents,  
1 = cluster of documents

MetaCarta Geographic Text Search Beta v1.0 - please send comments to [info@metacarta.com](mailto:info@metacarta.com)

104

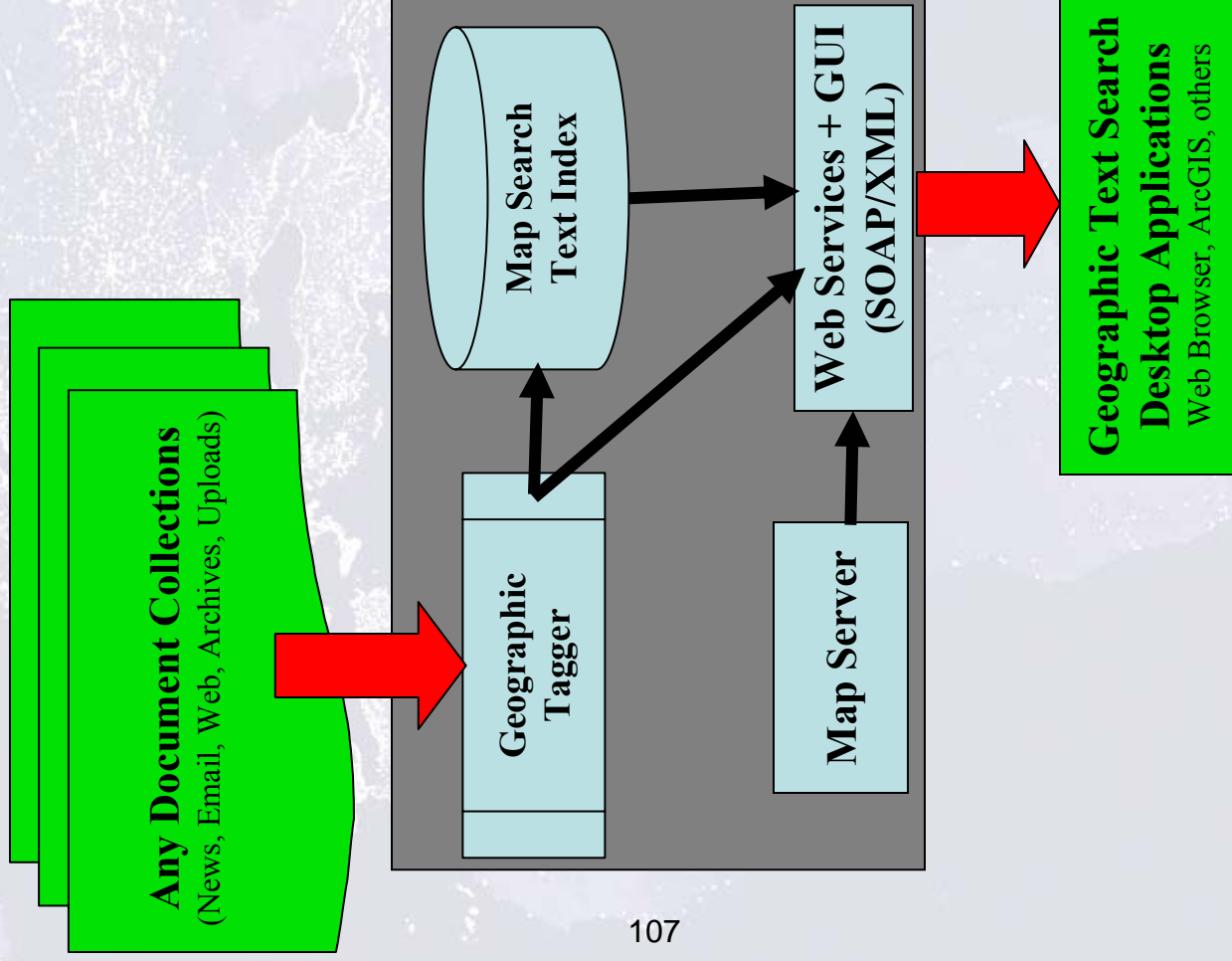








# MetaCarta™ GTS Appliance



Complete plug-n-play system with

- Browser-based GUI
  - Web Service for desktop applications.
  - Mapserver with basic vector data.
- (Ask us about connecting to other maps/imagery.)

-Or -

Geographic Tagger module for  
custom integration (SOAP/XML)



## Out of the box in three steps:

1. Mount your archive data disk.
2. One command launches geo-tagging and indexing.
3. Direct browser to <http://metacarta.mynetwork.ic.gov/>

# Several Modes of Analysis

- Hotspots of intensity light up for pattern detection.
- Reduce cutoff for high recall on sensitive searches.
- Correlate reports and overhead imagery for operations.
- And much more...

# Hotspots become visible.

© MetaCarta Search: site:mitre.org osama bin laden

Address: [http://demo.metacarta.com/demo.pl?zoom\\_level=9&lon=60.096618&lat=28.985508&iconlink=zoom&keywords=site%3Amitre.org+osama+bin+laden](http://demo.metacarta.com/demo.pl?zoom_level=9&lon=60.096618&lat=28.985508&iconlink=zoom&keywords=site%3Amitre.org+osama+bin+laden) go

Back Forward Stop Refresh Home AutoFill Print Mail

**META CARTA** search



Enter search terms [Tips and Instructions](#)  
site:mitre.org osama bin laden

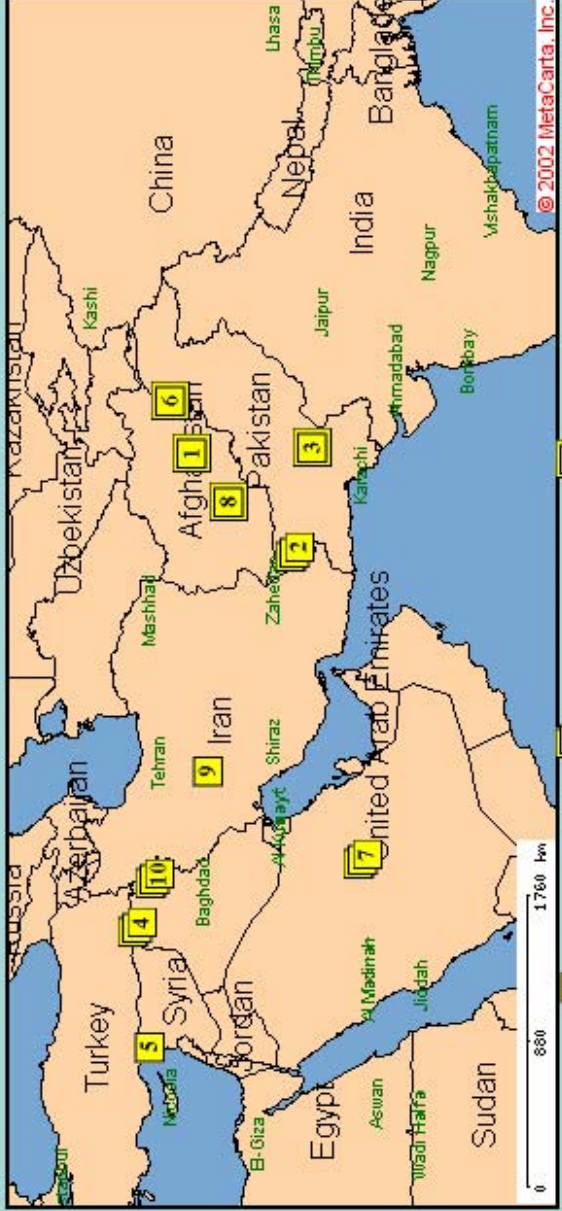
Place/Street address Region/City Country/State  
77 Mass Ave Cambridge MA

Submit

[Reset search](#) [Reset map](#)

Clicking yellow icons:  
☒ zooms map in  
☐ views documents

Zoom In  Zoom Out 



© 2002 MetaCarta, Inc.

**1** = single document, **1** = stack of documents, **1** = cluster of documents

MetaCarta Geographic Text Search Beta v1.0 - please send comments to [info@metacarta.com](mailto:info@metacarta.com)

Search results (10 clusters of 811 results, page 1) [Next ->](#)

**1** 58% [Bin Laden fled to Iran, cook says](#)

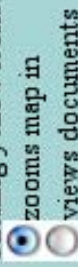
[to Iran, cook says **GHAZNI**, **AFGHANISTAN** - ]...important leaders were in that convoy including **bin Laden** and the Egyptian doctor [ Aymen ]...I haven't seen any evidence that [



For example,  
let's look at Lackland AFB.



Clicking yellow icons:



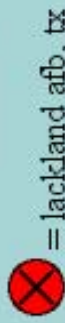
Enter search terms [Tips and Instructions](#)

Submit

Place/Street address Region/City Country/State

77 Mass Ave Cambridge

[Reset search](#)  
[Reset map](#)



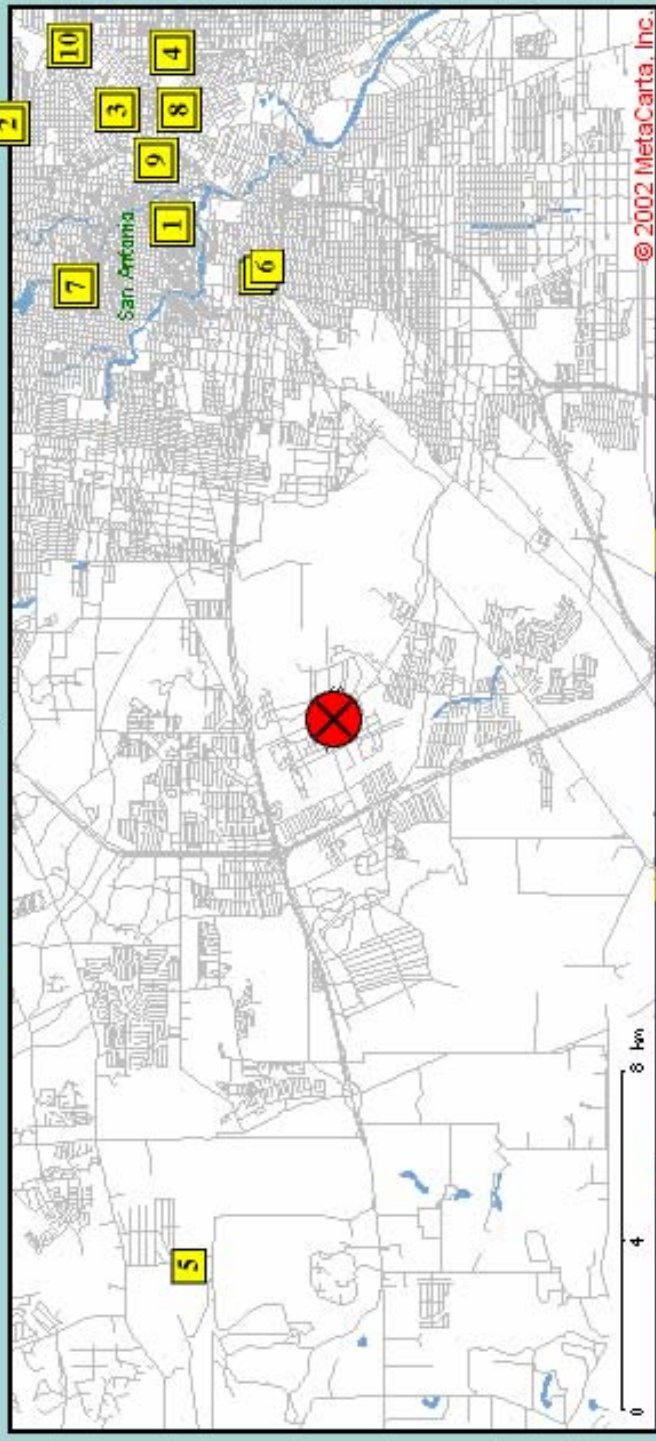
= lackland afb, tx



Zoom In



Zoom Out



**1** = single document, **1** = stack of documents; **1** = cluster of documents

MetaCarta Geographic Text Search Beta v1.0 - please send comments to [info@metacarta.com](mailto:info@metacarta.com)

Search results (10 clusters of 5215 results, page 1) [Next ->](#)

**1** 84% [Contact Us](#)

[1327 Guadalupe Street San Antonio Texas] Contact Us ; 1327 Guadalupe Street San Antonio Texas ;  
78207 TX ; (210) 222-2151 Fax: (210) 222-4405 Email: [info@metacarta.com](mailto:info@metacarta.com)





Enter search terms [Tips and Instructions](#)

air force

Submit

Clicking yellow icons:

☒ zooms map in

☐ views documents

Place/Street address

77 Mass Ave

Region/City

Cambridge

Country/State

MA

[Reset search](#)  
[Reset map](#)



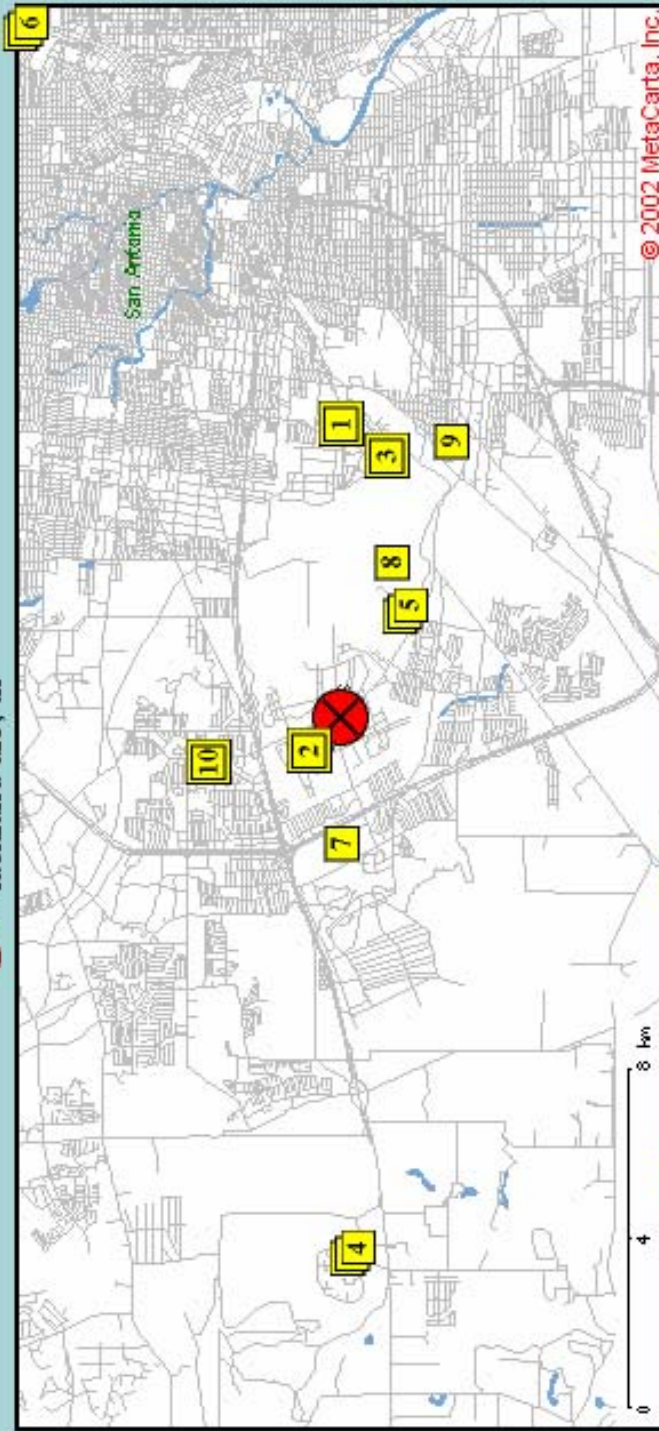
= lackland afb, tx



Zoom In



Zoom Out



**1** = single document; **1** = stack of documents; **1** = cluster of documents

MetaCarta Geographic Text Search Beta v1.0 - please send comments to [info@metacarta.com](mailto:info@metacarta.com)

Search results (10 clusters of 406 results, page 1) [Next ->](#)

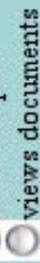
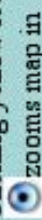
**1** 90% [Air Force News Agency](#)

[203 Norton Street, San Antonio, TX 78226-1848]...October 1983 the Air Force AFRTS Directorate

was designated Air Force...and became an...member...of the Air Force



Clicking yellow icons:



Enter search terms [Tips and Instructions](#)

air force

Submit

Place/Street address

77 Mass Ave

Region/City

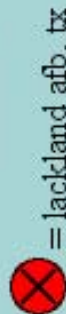
Cambridge

Country/State

MA

[Reset search](#)

[Reset map](#)



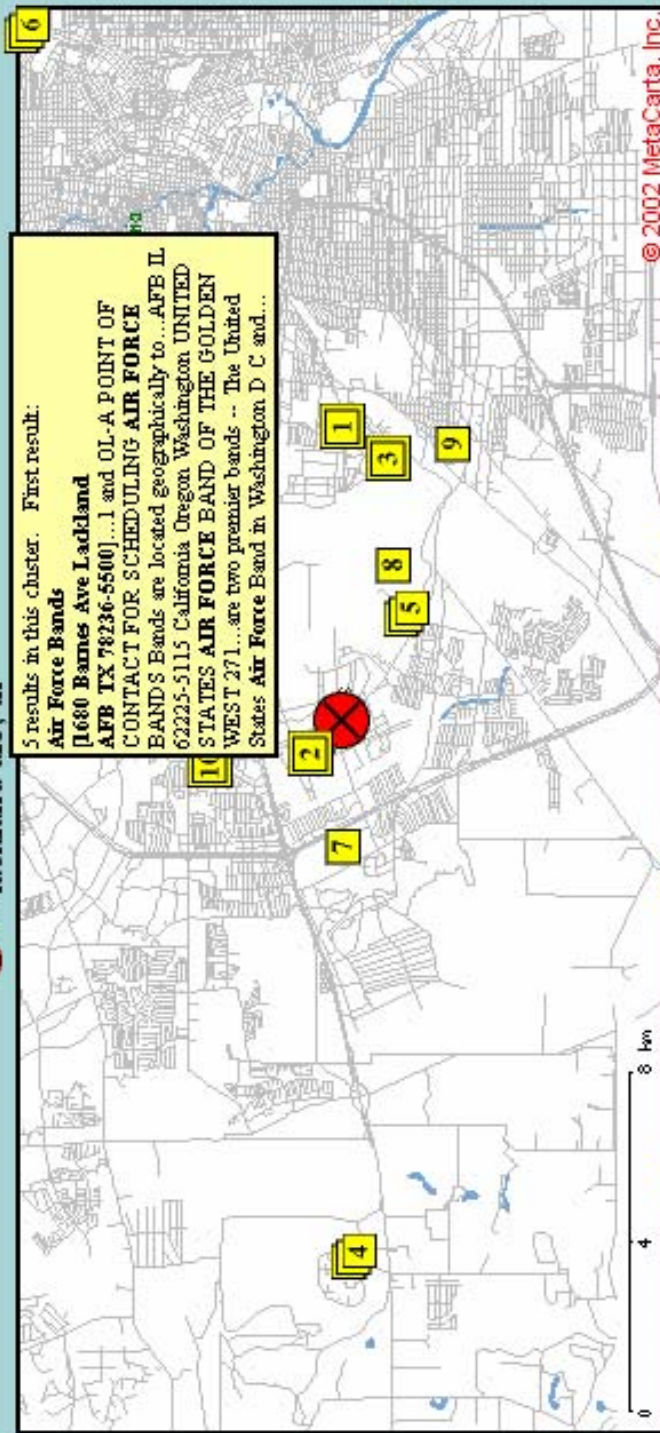
= lackland afb, tx



Zoom In



Zoom Out



**1** = single document; **1** = stack of documents; **1** = cluster of documents

MetaCarta Geographic Text Search Beta v1.0 - please send comments to [info@metacarta.com](mailto:info@metacarta.com)

Search results (10 clusters of 406 results, page 1) [Next ->](#)

**1** 90% [Air Force News Agency](#)

[203 Norton Street, San Antonio, TX 78226-1848]...October 1983 the Air Force AFRTS Directorate

was designated Air Force Broadcasting Service and became an integral part of the Air Force





# FACT SHEET

USAF Fact Sheet

## Air Force Bands

### Mission

Air Force bands support the global Air Force mission in war and peace by fostering our national heritage and by providing professional musical products and services for official military, recruiting and community relations events. They provide military and patriotic music for official military and government activities such as ceremonies, formations and parades. They also provide an essential element in maintaining troop morale, retention and recruiting efforts.

Bands play a key role in cultivating positive relations with many communities interacting with Air Force units. In addition, bands enhance public relations efforts with those communities outside the local areas of our military installations. In public concerts, parades and ceremonies, Air Force bands keep alive and enrich American musical heritage while projecting the Air Force image.

### Organization

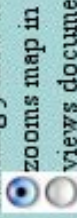
Air Force bands are classified as premier bands and regional bands. There are two premier bands -- The United States Air Force Band in Washington, D.C., and the United States Air Force Band of the Rockies in Colorado Springs, Colo. Ten regional bands are found at eight locations in the continental United States and operate from four locations overseas (Germany, Japan, Alaska and Hawaii). In addition, there are 14 Air National Guard bands at



Let's zoom in for a closer look.



Clicking yellow icons:

Enter search terms [Tips and Instructions](#)

air force

Submit

Place/Street address

77 Mass Ave

Region/City

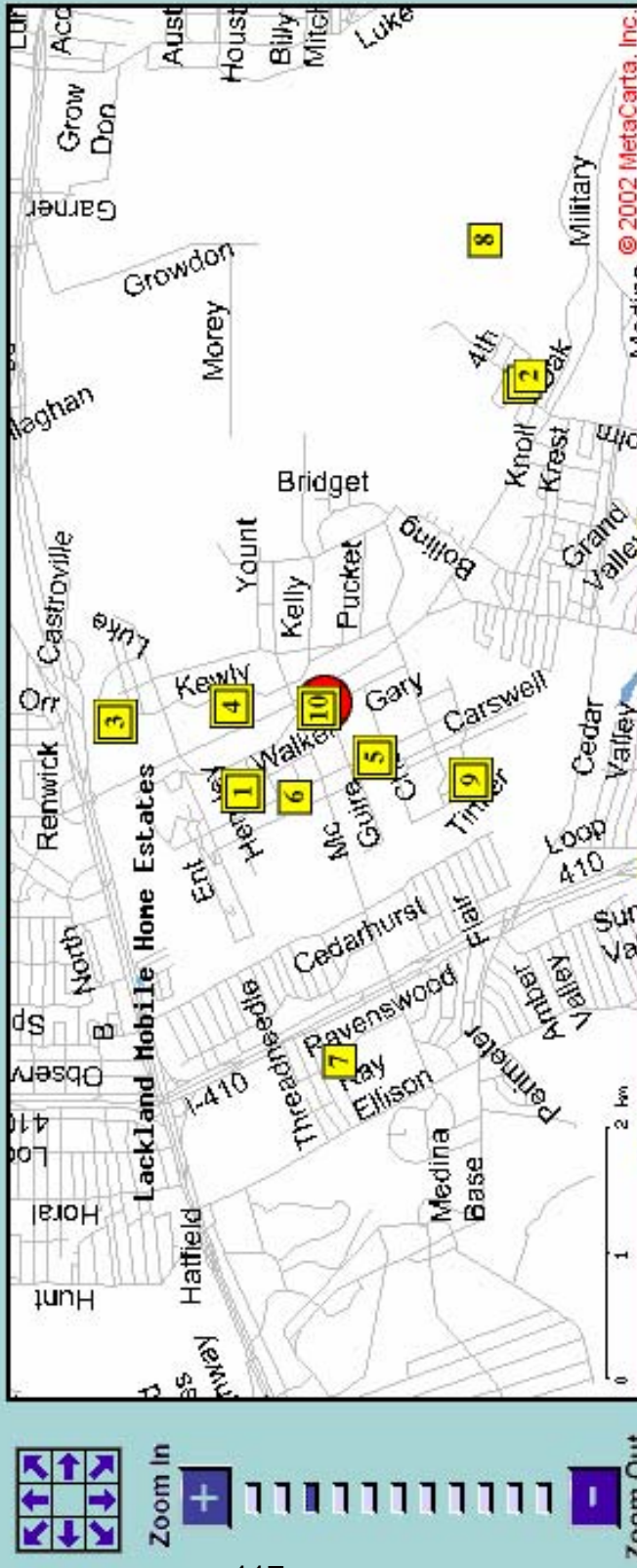
Cambridge

Country/State

MA

[Reset search](#)[Reset map](#)

= lackland afb, tx



**1** = single document, **1** = stack of documents, **1** = cluster of documents

MetaCarta Geographic Text Search Beta v1.0 - please send comments to [info@metacarta.com](mailto:info@metacarta.com)

Search results (10 clusters of 101 results, page 1) [Next ->](#)**1** 84% [Air Force Bands](#)

[1680 Barnes Ave Lackland AFB TX 78236-5500]... 1 and OL-A POINT OF CONTACT FOR SCHEDULING AIR FORCE BANDS Bands are located geographically to AFB TX 78236-5500

# Coming soon... Search by time range.



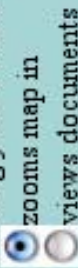


Enter search terms [Tips and Instructions](#)

date:170302 air force

Submit

Clicking yellow icons:



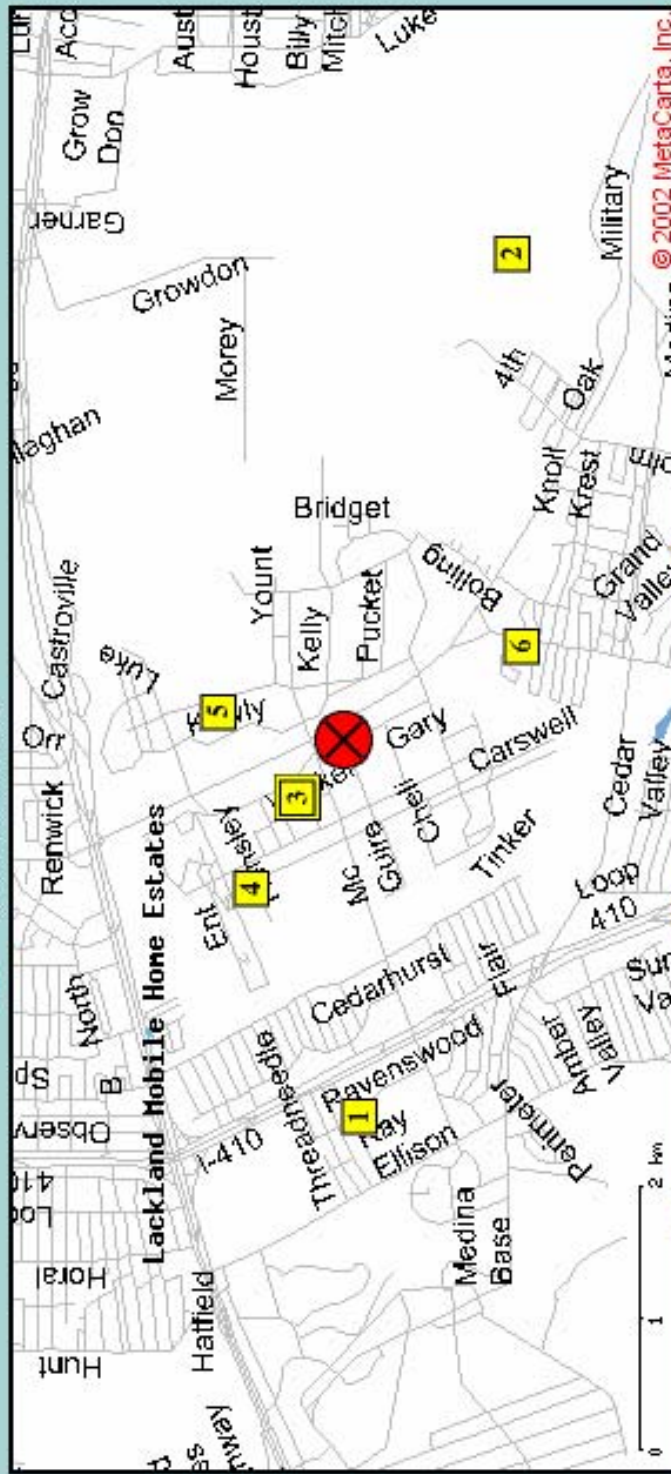
Place/Street address Region/City Country/State

77 Mass Ave Cambridge MA

[Reset search](#)  
[Reset map](#)



= lackland afb, tx



**1** = single document, **1** = stack of documents, **1** = cluster of documents

MetaCarta Geographic Text Search Beta v1.0 - please send comments to [info@metacarta.com](mailto:info@metacarta.com)

Search results (6 clusters of 15 results, page 1) [Next ->](#)

**1** 61% [Sub-Section 9.8.10 - Other - Housing/Homes](#)

[4917 Ravenswood Dr. San Antonio, TX 78227] is available to widows/dependents of retired



Enter search terms [Tips and Instructions](#)

Submit

zulu:0800-0930 date:170302 air force

[Reset search](#)  
[Reset map](#)

Place/Street address Region/City Country/State  
77 Mass Ave Cambridge MA

Clicking yellow icons:  
zooms map in  
views documents



Zoom In

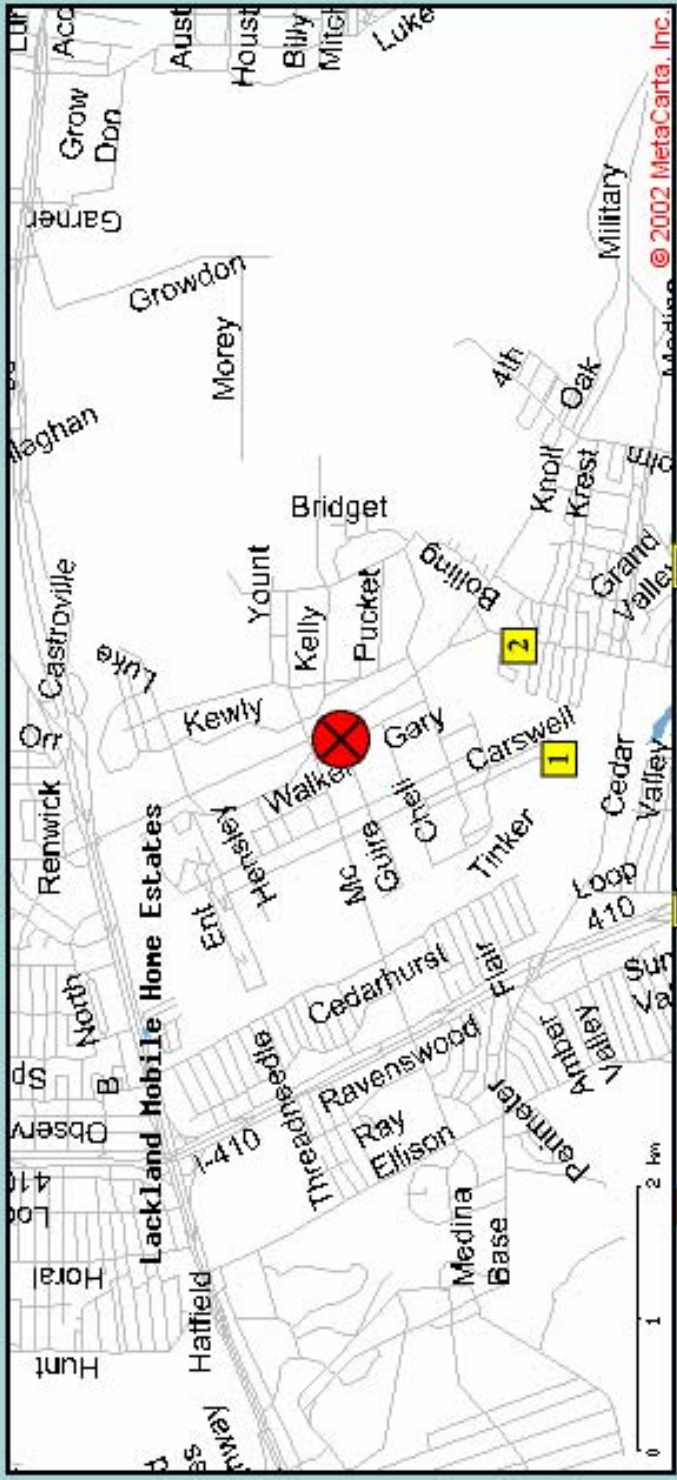


Zoom Out

120

Zoom Out

= lackland afb, tx



1 = single document, 1 = stack of documents; 1 = cluster of documents

MetaCarta Geographic Text Search Beta v1.0 - please send comments to [info@metacarta.com](mailto:info@metacarta.com)

Search results (2 clusters of 2 results, page 1) [Next ->](#)

1 22% [Call for Papers](#)





# Geographic Text Search

Where can MetaCarta take you?

Call about deployments, upcoming features,  
custom development, and integration.

617-661-6382

[john.frank@metacarta.com](mailto:john.frank@metacarta.com)